# APPLICATION OF FUZZY LOGIC FOR COLLISION AVOIDANCE IN NAVIGATION AND MANIPULATION IN DYNAMIC ENVIRONMENTS

*A Thesis Submitted*
in Partial Fulfilment of the Requirements
for the Degree of
**Doctor of Philosophy**

*by*
SHRIMAN ANUPAM BAGCHI

*to the*
Department of Mechanical Engineering
INDIAN INSTITUTE OF TECHNOLOGY
Kanpur 208 016, INDIA
**November, 1991**

1 9 OCT 1993 /ME

CENTRAL LIBRARY
I.I.T. KANPUR

Acc. No. A. 116567

ME-1991-D-BAG-APP

# SYNOPSIS

| | | |
|---|---|---|
| Name of Student | : | Shriman Anupam Bagchi |
| Roll Number | : | 8620563 |
| Degree for which submitted | : | Doctor of Philosophy |
| Department | : | Department of Mechanical Engineering |
| Thesis Title | : | APPLICATION OF FUZZY LOGIC FOR COLLISION AVOIDANCE IN NAVIGATION AND MANIPULATION IN DYNAMIC ENVIRONMENTS |
| Names of Thesis Supervisors | : | (1) Amitabha Ghosh<br>(2) Himanshu Hatwal |
| Month and Year of submission | : | November, 1991 |

This thesis addresses the problem of collision avoidance of bodies in an unknown environment. It is assumed that an approximate perceived world model, based only on locally available instantaneous sensory information, is available, but its future time history is not known *apriori.* The endeavour is to evolve a solution strategy that (*i*) works when modelling the environment is difficult due to inherent uncertainties; (*ii*) is robust enough to take care of modelling inaccuracies; (*iii*) is flexible to modification to suit different problems; and (*iv*) is fast enough to work in real-time. To serve the above goals, **fuzzy logic** has been used in this work. Collision avoidance strategies are cast in the form of *fuzzy rules* which are based on intuition and can be changed easily to suit different problems. The collection of fuzzy rules forms the **fuzzy controller** which is used to make on-line decisions. The inputs to the fuzzy controller involve quantities like "perceived distance of the external body" and its "perceived angle of approach" while the output quantity is the "direction in which the body must move" to avoid collision. It has been shown that the above methodology can be effectively applied to prevent collision of bodies moving without prior knowledge of each other. The applicability of the algorithm has been demonstrated by considering the following three problems:

1. Collision avoidance for bodies moving on a plane.
2. Collision avoidance for a planar manipulator in a dynamic environment.
3. Collision avoidance for a 3-D manipulator in a dynamic environment.

POST GRADUATE OFFICE
Submitted on 19/11/91

# CERTIFICATE

It is to certify that the work contained in this thesis entitled ''APPLICATION OF FUZZY LOGIC FOR COLLISION AVOIDANCE IN NAVIGATION AND MANIPULATION IN DYNAMIC ENVIRONMENTS'' has been carried out under our supervision, and that this work has not been submitted elsewhere for the award of a degree.

[illegible] [illegible]

AMITABHA GHOSH HIMANSHU HATWAL

Department of Mechanical Enginering
I.I.T. Kanpur

November, 1991

THIS WORK IS DEDICATED TO

MY FAMILY MEMBERS

# Acknowledgements

First of all I wish to thank my supervisors, Dr. Himanshu Hatwal and Prof. Amitabha Ghosh for their whole hearted support towards my work. The extempore comments made by them had given me the seed thought which fructified after five years. I thank Dr. Hatwal for the precious time he has spent on studying the first drafts of this thesis and suggesting changes which gave a new direction to my work and vastly improved the style of presentation. He almost never got tired sitting till late hours of night, reading and revising drafts of my thesis.

Next I wish to thank all my family members who have given their unstinted cooperation while I was away from home. Although I had never been able to impress upon them the validity of working for a doctoral degree, they suffered silently due to what they thought was an expression of my impractical nature. I am highly indebted to my wife, Sumona, who stepped into my life while I was mid-way through my work. She seemed to understand well the uncertainty surrounding the completion of my work, and waited patiently to see the end of it.

Thanks are also due to Dr. Tapan Bagchi for the numerous discussions I had with him regarding reliability analysis of the algorithms. His interrogative looks while we crossed each other, has always been a driving force behind my work. I must also express my thanks to Mrs. Mita Bagchi for the unexpected dinner parties I was often invited to, sometimes for valid reasons and at other times for practically no reason.

My stay at IIT Kanpur has been spiced by the presence of many friends who have provided the much needed diversion. Special mention must be made of Samiran Mandal, Subrata Saha, Debashish Bhattacharjee, Praveen Bhatia, Laxmi Prasad, Ngo-Sy-Loc and Sukanta Kundu with whom I spent most of my leisure time and sometimes productive time. Last but not the least I express my sincere gratitude toward the laboratory staff, Vivek Shukla, Mr. Susmit Sen, Dr. S.R. Pandian, Mrs. Anjali Kulkarni and Ms. Sushma Sinha who have given me company during my work. The Centre for Robotics at IIT Kanpur has provided me an ideal working atmosphere. Use of the facilities at the Centre has saved me the trouble of undergoing the bureaucratic hurdles at other places.

[illegible]
19/11/91

November 1991 S. ANUPAM BAGCHI

# Contents

# List of Figures

# List of Tables

# List of Special Symbols

| | | |
|---|---|---|
| $A^{\alpha}$ | : | $\alpha$-level of fuzzy set $A$ |
| $a_i$ | : | length of link $L_i$ in 2-D problem |
| $A_{\alpha}$ | : | $\alpha$-cut of fuzzy set $A$ |
| $a_{\max}$ | : | linear acceleration limit of a body |
| $\vec{\mathbf{a}}$ | : | approach vector of hand |
| $\vec{a}_P$ | : | acceleration vector of $\mathbf{P}$ |
| B | : | base of the manipulator |
| $B_i$ | : | body on a plane |
| $b_i$ | : | width of link $L_i$ in 2-D problem |
| $B_i^G$ | : | goal position of body $B_i$ |
| $B_i^S$ | : | starting position of body $B_i$ |
| $B_i^{Gp}$ | : | intermediate goal position of body $B_i$ |
| $Card(A)$ | : | cardinality of fuzzy set $A$ |
| $CON$ | : | concentration operation |
| $C_{i,Lh}$ , $C_{i,Lt}$ | : | sensors on left side of link |
| $C_{i,Rh}$ , $C_{i,Rt}$ | : | sensors on right side of link |
| $\vec{c}$ | : | vector defining a ray from sensor $C$ |
| $\vec{C}_{Esc}$ | : | escape vector acting at sensor $C$ |
| $DIL$ | : | dilation operation |
| $d_{P_1,ij}$ | : | the distance from observation point $P_{1,i}$ |

| | | |
|---|---|---|
| | | to point obstacle $O_j$ |
| $\vec{d}_0$ , $\vec{d}_1$ | : | measured distances |
| $E$ | : | end-effector position of the manipulator |
| E | : | 'elbow' of PUMA manipulator |
| $E^G$ | : | goal point of the end-effector |
| $E^S$ | : | starting point of the end-effector |
| $E^{Gp}$ | : | intermediate goal position of the end-effector |
| $E^{[1]}, E^{[2]}$ | : | positions of the end-effector |
| $\vec{e}_i$ | : | the effective escape vector acting at $t_i$ |
| F | : | 'finger' of PUMA manipulator |
| $F'$ , $F''$ | : | intermediate positions of the finger $F$ |
| $\vec{F}_{Eff}$ | : | effective escape vector at finger $F$ |
| $\vec{F}_{Esc}$ | : | escape vector acting at finger $F$ |
| $\vec{g}$ | : | direction of projection |
| $\mathrm{H}_D$ | : | largest measurable distance |
| $Hgt(A)$ | : | height of fuzzy set $A$ |
| $h_i$ | : | point representing head of link $\mathrm{L}_i$ |
| ${}^{i-1}[\mathbf{A}]_i$ | : | the transformation matrix for link $i$ |
| inf | : | infimum operation |
| $J$ | : | Jacobian matrix |
| $K\acute{e}r(A)$ | : | kernel of fuzzy set $A$ |
| $\ell$ | : | level of prediction |
| $L$ | : | Lagrangian function |
| $\mathcal{L}$ | : | lattice |
| $\mathrm{L}_i$ | : | link $i$ of a manipulator |
| $m$ | : | number of obstacles |
| $M(\mathcal{A})$ | : | meaning of syntagm $\mathcal{A}$ |
| $\mathcal{M}$ | : | semantic rule |
| $n$ | : | number of links or bodies |
| $\mathsf{N}$ | : | set of Natural numbers |
| $N_A$ | : | number of elements in $U_A$ |
| $N_D$ | : | nmber of elements in $U_D$ |
| $\vec{\mathrm{n}}$ | : | normal vector of hand |
| $\vec{n}$, $\vec{m}$ | : | normal vectors to a plane |
| $O^{[1]}, O^{[2]}$ | : | positions of obstacles |
| $O_j$ | : | the $j^{th}$ point obstacle |
| $P_{1,i}$ | : | the first observation point on link $i$ |
| $P_{2,i}$ | : | the second observation point on link $i$ |

| | | |
|---|---|---|
| $\vec{\mathrm{p}}$ | : | position vector of hand |
| $\mathbf{P}$ | : | the primary body |
| $\vec{p}_S^{future}$ | : | future position of secondary body |
| $\vec{p}_S^{past}$ | : | past position of secondary body |
| $\vec{p}_S^{present}$ | : | present position of secondary body |
| $r_n$ | : | number of rules in the fuzzy controller |
| $\mathsf{R}$ | : | set of real numbers |
| $Rot^{[p]}$ | : | rotation operator on a fuzzy number over $U_A$ |
| $\vec{R}$ | : | vector defining a path |
| $s$ | : | arc length |
| sup | : | supremum operation |
| $\mathsf{S}$ | : | 'shoulder' of PUMA manipulator |
| $\vec{\mathrm{s}}$ | : | sliding vector of hand |
| $\mathsf{S}$ | : | the secondary body |
| $Supp(A)$ | : | support of fuzzy set $A$ |
| $S^+$ , $S^-$ | : | fuzzy set with special chaaracteristics |
| $s_1$ , $s_2$ | : | proportion factors used for sensor placement |
| $t_0$ , $t_1$ | : | instants of time |
| $t_i$ | : | point representing tail of link $\mathrm{L}_i$ |
| $t_{i,new}$ | : | intermediate position of $t_i$ used for reorientation |
| $\mathbf{T}$ | : | the arm matrix |
| $\vec{T}$ | : | tangent to a path |
| $\mathcal{T}(\mathcal{X})$ | : | term set of variable $\mathcal{X}$ |
| $U$ | : | universe |
| $U_A$ | : | Universe of Angle |
| $U_D$ | : | Universe of Distance |
| $V$ | : | universal set |
| $v_{\max}$ | : | linear velocity limit of a body |
| $\mathbf{V}_{SP}$ | : | fuzzified relative velocity over $U_A$ |
| $\vec{v}_1$, $\vec{v}_2, \ldots$ | : | unit vectors |
| $\vec{v}^*$ | : | nominal velocity of a body |
| $\vec{v}_P$ | : | velocity of the primary body |
| $\vec{v}_S$ | : | velocity of the secondary body |
| $\vec{v}_{B_i}$ | : | velocity vector of body $B_i$ |
| $\vec{v}_{B_i,Esc}$ | : | the escape vector acting at body $B_i$ |
| $\vec{v}_{P,f}$ | : | final velocity of $\mathbf{P}$ |
| $\vec{v}_{P,i}$ | : | initial velocity of $\mathbf{P}$ |

| | | |
|---|---|---|
| $\vec{v}_{P,\,Esc,\,S}$ | : | absolute escape vector on **P** due to **S** in the base frame |
| $\vec{v}^{R}_{P,\,Esc,\,S}$ | : | absolute escape vector on **P** due to **S** in the rotated frame |
| $\vec{v}_{P,\,Esc}$ | : | net escape vector on **P** |
| $\vec{v}^{R}_{P,\,relEsc,\,S}$ | : | relative escape vector on **P** due to **S** in the rotated frame |
| $\vec{v}_{P_{1},i}$ | : | the escape vector acting at $P_{1,i}$ |
| $\vec{v}_{SP}$ | : | relative velocity of **S** w.r.t. **P** |
| W | : | 'wrist' of PUMA manipulator |
| $W'$ , $W''$ | : | intermediate positions of the wrist $W$ |
| $\vec{W}_{Eff}$ | : | effective escape vector acting at wrist $W$ |
| $\vec{W}_{Esc}$ | : | escape vector acting at wrist $W$ |
| $\vec{w}_{h_i}$ | : | the apportioned escape vector acting at $h_i$ on same link |
| $\vec{w}_{t_i}$ | : | the apportioned escape vector acting at $t_i$ on same link |
| $X_0$-$Y_0$ | : | the base coordinate system |
| $X_0^R$-$Y_0^R$ | : | the rotated coordinate system |
| $X_L$-$Y_L$ | : | the link coordinate system |
| $X$, $Y$ | : | classes |
| $\mathcal{X}$ | : | name of a fuzzy variable |
| $(x_F, y_F, z_F)$ | : | position coordinates of the end-effector with respect to $X_0$-$Y_0$ |
| $\vec{z}_{h_i}$ | : | the apportioned escape vector acting at $h_i$ due to an adjacent link |
| $\vec{z}_{t_i}$ | : | the apportioned escape vector acting at $t_i$ due to an adjacent link |
| $\alpha$ | : | beam angle of an ultrasonic sensor |
| $\alpha_{i,max}$ | : | the maximum limit of joint acceleration $\ddot{\theta}_i$ |
| $\ddot{\theta}_i$ | : | the angular acceleration of joint $i$ |
| $\Delta t$ | : | time interval |
| $\Delta x$ | : | incremental $x$ displacement in base frame $X_0$-$Y_0$ |
| $\Delta y$ | : | incremental $y$ displacement in base frame $X_0$-$Y_0$ |
| $\Delta_1$ , $\Delta_2$ ... | : | areas of polygons |
| $\Delta_{shadow}$ | : | the total shadow area |
| $\dot{\theta}_i$ | : | the angular velocity of joint $i$ |
| $\lambda_1$ , $\lambda_2$ | : | slack variables used for optimization |
| $\mu$ | : | linguistic modifier |

| | | |
|---|---|---|
| $\omega$ | : | the angular velocity of a body |
| $\omega_{i,max}$ | : | the maximum limit of joint velocity $\dot{\theta}_i$ |
| $\omega_{\max}$ | : | angular velocity limit of a body |
| $\phi$ , $\psi$ | : | angles |
| $\phi_{P_1,ij}$ | : | the angle from observation point $P_{1,i}$ to point obstacle $O_j$ |
| $\Theta$ , $\Theta^R$ | : | fuzzy quantities over $U_A$ |
| $\theta_i$ | : | the value of joint $i$ |
| $\theta_{SP}$ | : | direction of vector $\vec{v}_{SP}$ |
| $\varrho$ , $v$ | : | apportioning factors |
| $\widehat{f}$ | : | the front vector of a body |
| $(\theta_i)_{lower}$ | : | the lower limit of joint $i$ |
| $(\theta_i)_{upper}$ | : | the upper limit of joint $i$ |
| $\boxtimes$ | : | symbol for bold intersection |
| $\cap$ , $\bigcap$ | : | symbol for intersection |
| $\circ$ | : | symbol for strong composition |
| $\cup$ , $\bigcup$ | : | symbol for union |
| $\circledRightarrow$ | : | symbol for residuum on fuzzy sets |
| $\in$ | : | symbol meaning 'is an element of' |
| $\leftrightarrow$ | : | symbol for biresiduum |
| $\otimes$ , $\rightarrow$ | : | multiplication and residuum operation on fuzzy sets |
| $\sqsubseteq$ | : | symbol meaning 'is a fuzzy set in universe' |
| $\times$ | : | symbol for cartesian product |
| $\vee$ | : | 'or' operation |
| $\overset{\circ}{\times}$ | : | symbol for weak composition |
| $\wedge$ | : | 'and' operation |

*Between the idea*
*And the reality,*
*Between the motion*
*And the act,*
*Falls the Shadow.*
– T. S. ELIOT

# Chapter 1

# Introduction

## 1.1 Introduction

Collision avoidance is an area of research having a number of applications in robotics, specially in designing autonomous systems. Its primary usage is in motion planning of bodies amongst obstacles. The initial research thrust was for motion planning in a completely *known* environment. Recently the emphasis is to work in an *unknown* environment using sensory perception. In simple terms, motion planning is the automatic coordination of moving bodies such that they do not collide between themselves while maintaining the kinematic and dynamic constraints. The main areas of application for motion planning algorithms are, in navigation and generating collision free trajectories of manipulator arms.

Navigation is the fundamental requirement of autonomous mobile vehicles. Building a completely autonomous system capable of navigating through an unknown and dynamic environment is quite a formidable task. The issues involved in making a completely autonomous vehicle are sensing, world model building, decision making, control and motion planning among many others. In this work, we focus attention on the motion planning strategy to be adopted by an autonomous vehicle if it were to navigate through a space with other moving vehicles.

Manipulation problems for robots can be thought of as an extension of motion planning concepts developed for navigation of vehicles. In general, a planning problem in navigation and manipulation can be reduced to a search within some space. For navigation problems, this space is the plane of motion, while for manipulation problems this is the workspace of the manipulator. It is thus important to realize that a motion planning problem can be thought of as a general class of problems dealing with both navigation and manipulation within a workspace. Thus for convenience, the term 'robot' has been used to refer to both autonomous vehicles and manipulators.

Most conventional programming methods for robot manipulators and autonomous vehicles give direct commands to the system which are meant to be executed without interaction with the environment. These methods provide the fastest means of programming when the environment is perfectly known and is not liable to change abruptly. Such kinds of environments are called *structured environments*. Most structured environments are artificially created indoors, for example, in a factory or a laboratory. This situation is far from practical for outdoor settings, because most natural environments are unknown beforehand and are liable to change suddenly. These environments are called *unstructured environments*. Working in unstructured environments poses a bigger challenge to developers of robotics because of the various uncertainties involved. It is paramount to realize that systems must now be endowed with the capability of *interacting* with the environment and *adapting* to changes i.e., the ability to change the operating parameters of the system to suit the present status of the environment.

Some advantages of granting more autonomy to robotic systems are as follows. First, that systems need not be re-programmed every time small modifications are required because they are made capable of handling minor uncertainties encountered during execution. Secondly, a requirement that gains further ground is the development of, what is called *task-level language*. In a task-level language low-level robot commands are not required to be explicitly written. Programming a robot is the mere specification of commands at the *task-level*, for example, "Move to position $A$ avoiding obstacles." The actual move sequences to accomplish this task are generated automatically by the system without human intervention. The other advantage of autonomy in the factory environment, is that it will reduce down time needed during robot re-programming. Autonomous systems are also desirable where humans cannot accompany the system for interactive handling in remote or inaccessible areas like inner space, deep underwater or in nuclear environments.

## 1.2 Literature Review

### 1.2.1 General aspects of motion planning

One of the primary concerns of a planning problem, is to be able to handle the *interactions* between objects in the work environment and the robot. In some cases the objective is to avoid a collision between two approaching bodies whereas, in other cases it requires careful contact between two mating parts, such as in assembly operations. Conventional robot planning programs embody assumptions about the geometry of the physical environment (e.g., the locations of parts), the semantics of the robot commands (e.g., the robot is position controlled), and the mechanics of the task (e.g., the masses and coefficients of friction of objects to be manipulated) [87]. Difficulty arises when any of these assumptions are violated (e.g., when one of the parts is displaced from its expected position). Under such circumstances, these programs might fail to achieve the intended task.

Based on the amount of uncertainty in the planner's knowledge of the task environment, planning problems can be categorized as follows [87]:

1. Planning with complete prior knowledge.

2. Planning with incomplete planning-time knowledge but complete execution -time knowledge.

3. Planning with uncertainty.

## Planning with complete prior knowledge

The planning assumption embodied in such problems is that any variation in layout, part locations and shape, and manipulator behaviour are negligible. The problems faced in planning such tasks are exclusively geometric. Information about the world is assumed to be completely known beforehand.

There are two related, but distinct, lines of research in collision-free motion planning. One is focused on approximate numerical algorithms [4,12,39,40,41,83] and the other is focused on exact combinatorial algorithms [113,112,111,115,114, 134]. Research on approximate algorithms has lead to practical algorithms, while research on combinatorial algorithms has generated several new approaches to motion planning and refined our understanding of the inherent complexity of the problem.

Both approaches to motion planning can be characterized as search strategies in the *configuration space* [83] of the robot. *Configuration space* is the parameter space of positions. The obstacles in this space represent values of parameters that cause collisions. Subsets of the robot's configuration space that are free of collisions, represent the *free space*. Path-planning is then the problem of finding a path, within the free space, that connects the initial and final configurations. If the initial and final configurations are not on the same path-connected component of free space, then a feasible path is not possible. An advantage of computing the path-connectivity of free space is that this computation can be performed once as a pre-processing step, and then requests for paths between query configurations can be processed without recomputing free space each time. Methods for testing path connectivity in free space for general path planning problems have been analyzed by Canny [18], and Schwartz and Sharir [113,112,111,115,114].

Methods of collision avoidance based on characterizing free space have been dealt by Udapa at Caltech [130], Lozano-Pérez at MIT [81], Lozano-Pérez and Wesley at IBM [79] and Brooks at MIT [13].

A free-space representation based on high-level descriptions, as *generalized cones*, of the empty corridors between obstacles was introduced by Brooks [13]. A *generalized cone* was conceptualized as the free space between a pair of obstacle walls, with the *spine* as the bisector of the cone. The collection of spines formed a lower dimensional representation of free space and represented the locus of points that were locally maximally distant to obstacles. In a subsequent paper [14] Brooks discussed extending this representation to include not only *highways* (i.e., generalized cones) but also *meadows* — convex areas of free space. These descriptions were used to compute motion constraints for convex objects moving along the centre of the corridors. The free space was represented as a *tree of polyhedral cells* at varying resolutions by Lozano-Pérez in [80,81]. Path searching was done by searching a graph whose nodes were cells in the free-space representation and whose links denoted overlap between the cells.

Work on finding efficient representations of configuration space and fast searching algorithms still continues. In [69] the free space concerning path planning was restricted to avoid executing unnecessary collision detections. For efficient searching, the configuration space was quantized into cells by placing a regular grid. In [138] an approach to cell decomposition based on 'constraint reformulation' was proposed. An algorithm for hierarchical search with a mechanism for recording failure was given which was claimed to be considerably faster than other planners based on the same approach.

Other representations of the free space employed the *Voronoi diagram* [134, 124,19] or the *Visibility graph* [79]. The Voronoi diagram represents the locus of points that are equidistant from at least two of the obstacle boundaries, while the Visibility graph connects those vertices of obstacles that can "see" each other, that is, that can be connected by a straight line without penetrating any of the obstacles (see Figure 1.1). A Voronoi diagram is a lower dimensional representation of the connectivity of free space. The information content of a map representation through the Voronoi diagram is essentially the shape and position of obstacles. This representation was shown to be useful for path-planning, since it lead to paths that were optimally distant from obstacles. Both the Voronoi diagram and the Visibility graph concepts are useful tools for path-planning since they encode the connectivity between free regions. Recently [133] a Constructive Solid Geometry (CSG) representation of robots working in an octree model of the free work space has been proposed that is useful for Computer-Aided-Design (CAD) of robotic cells. The main intention of this model is to characterize selected 'moveability'

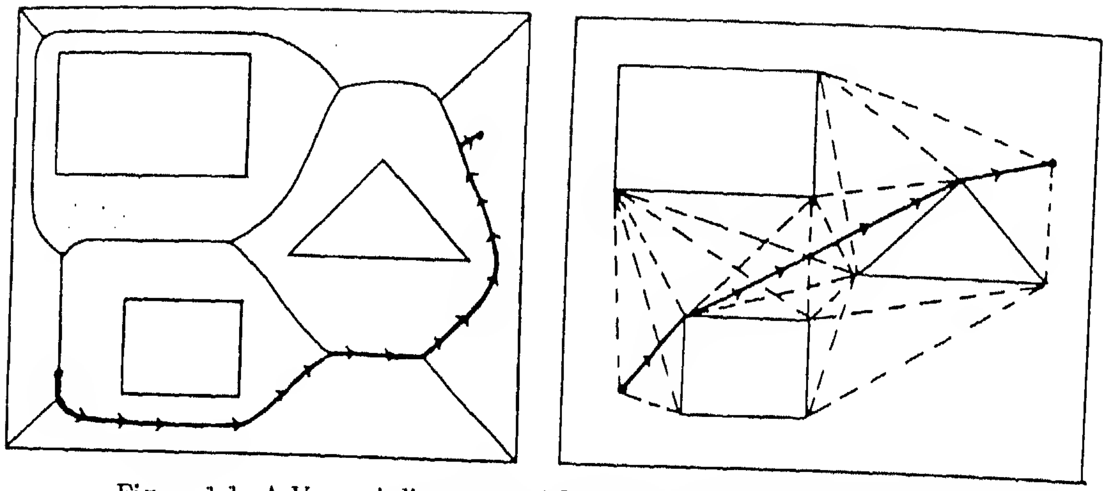


Figure 1.1: A Voronoi diagram and Visibility graph of a set of obstacles [87]

areas in the free work space.

The advantage of free-space methods is that they can guarantee to find a path if one exists within the free-space. Another advantage is that it is possible to search for *short* paths rather than simply finding the first path that is safe. A disadvantage is that the computation of the free-space may be expensive. Much research has been done on exploring methods of computing configuration space efficiently and proving bounds on the complexity of configuration space [50,66].

Another common proposal for obstacle avoidance is based on defining a *penalty function* on manipulator configurations that accounts for the presence of objects [65, 57,70]. This was emphasized by Le Maitre and Khatib [64,106] in which they suggested the relationship between cost function gradients and mechanical systems. Assuming that each obstacle is described as the zero level surface of a known scalar valued analytic function, $f(x, y, z) = 0$, Khatib formed a local inverse square potential law, thus creating a *potential field* around the obstacle. For avoiding collision between rigid manipulator *links* with obstacles, Khatib extended the idea of potential field in [65] where he proposed identifying a number of distinguished points on the manipulator body, and subjecting each of them to the potential field of the obstacles. This induced a potential function on the joint space of the manipulator. Joint limits were taken care of by adding obstacle potentials in the induced workspace. This function was called the FIRAS function (Force inducing an artificial repulsion from the surface) given in [65] as

$$F^*_{O,psp} = \begin{cases} \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{1.1}$$

where $\eta$ is a constant gain factor, $\rho$ is the shortest distance to the obstacle $O$, $\rho_0$ represents the limit distance of the potential field influence and $\partial\rho/\partial x$ represents the partial derivative vector of the distance from the PSP (*Point Subjected to the Potential*) on the manipulator body. The net motion of the robot results from the addition of these repelling forces and an attractive force pulling the robot toward the goal, subject to kinematic constraints.

The computation of penalty between non-spherical bodies (like links and arbitrary shaped obstacles) requires computing minimum distances. The obstacle shapes are generally modeled as super-quadric surfaces for this purpose [65,132] (see Figure 1.2). This method has been demonstrated to work well in practice when the environment is not densely populated with obstacles. A good use of potential functions is demonstrated for redundant robots with dynamic constraints by Barraquand et.al. [5] who also gave a heuristic method of searching the potential field for minima.

One disadvantage of using penalty functions is the "strictly local information that they provide for path searching" [87]. Pursuing the local minima of the penalty function can lead to situations where no further progress can be made. In these cases, the algorithm must choose a previous configuration where the search is to be resumed but in a different direction from the previous time. These *back-up* points are difficult to identify from local information. Another problem is that the use of analytic level surfaces to prescribe obstacle boundaries and neighbouring regions becomes difficult when the surface is complex, primarily because the relative locus of the concentric level surfaces may be poorly behaved. Although the existing methods of using penalty functions is more suited to work in configuration spaces where the exact state of the robot at any instant is represented by a single point, this method could also be applied to work in the task-space directly. However, in the task space the choice of an appropriate *PSP* on the manipula-

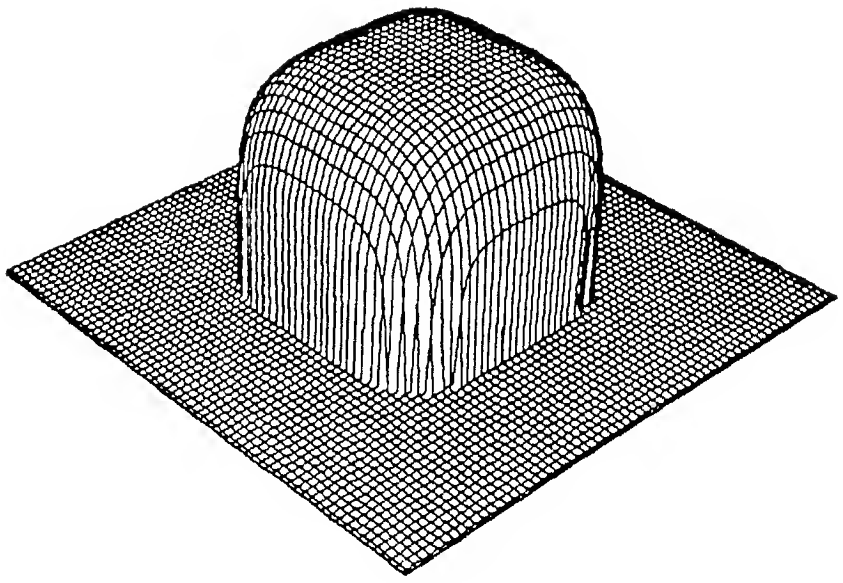


Figure 1.2: An example of an obstacle modeled through a super-quadric function [65]

tor body becomes crucial. While this guarantees that the particular points on the links will never collide with the obstacles, there is however no such guarantee for the entire link itself [87]. The above points suggest that the penalty function method might be combined profitably with a more global method of hypothesizing paths. Penalty function methods are more suitable for applications that require only small modifications to a known path.

Compared to the penalty function approach, the explicit free-space representation methods have an advantage when the goal is to bring two objects into contact, such as in grasp planning. This is so because they do not allow approximations of object shapes like the penalty function methods. The underlying assumption of penalty function methods, that objects shapes are not possibly important, does not hold when the objective is to generate plans to grasp objects or to approach the site of an assembly operation. The configuration space algorithms, in contrast, allow motion near to obstacles and are therefore applicable both to planning transfer motions, in choosing grasp configurations, and in planning assembly operations.

Among the exact combinatorial algorithms for path-planning, it is noteworthy to mention an article by Kedem and Sharir [63] which describes a planning algorithm for motion of a single vehicle. This algorithm is not only efficient and provably correct, but has also been implemented by the authors. The implemented system provides an "intelligent" robot that, using its attached vision system, can acquire a geometric description of the robot and its polygonal environment, and then, given a high-level motion command from the user, can plan a collision-free path (if one exists), and then go ahead and execute that motion.

More general path-planning problems involving motion constraints in the form of acceleration and speed bounds have been considered recently [102,17,62,60]. An initial study of planning motions when the moving object has an acceleration bound [102] described a method for finding a collision-free, acceleration bounded motion of a point on a line moving within oscillating barriers. Several algorithmic motion planning results [17,105] have been described for problems in which a speed bound has been imposed on an object moving in an environment containing moving obstacles. A heuristic approach to motion planning in the presence of moving obstacles was introduced by Kant and Zucker [62]. They first found a collision-free path in the environment in which the obstacles were fixed in their initial positions and then computed a motion of the object along the path so that the object avoided collisions with the moving obstacles while obeying a speed bound. Among the more recent contributions to this field, Fujimura and Samet [45], had converted the problem of 2-D motion planning for moving objects to a 3-D stationary problem by taking the $z$-axis as the time axis. A collision free path is now found in the 3-D extended stationary world of space-time by carefully avoiding unrealistic cases, like moving back in time. Recently [1] a rigourous analysis has been presented for determining how fast an object must be capable of moving for it to be able to reach a given position at a given time while avoiding moving obstacles. An algorithm working in $O(\log n)$ time has been presented which tells the lowest speed the moving object should follow to be able to reach the goal configuration without colliding with the obstacles. In another recent paper [116] motion planning of a manipulator in the presence of obstacles and dynamic constraints was attempted by integrating the path planning and trajectory planning problem together into an optimal control problem.

### Planning with incomplete planning-time knowledge but complete execution-time knowledge

Incomplete planning-time knowledge implies that the location of objects is not assumed to be completely known and some sensing action is used to reduce this uncertainty. By complete execution time knowledge it is assumed that once the presence of objects and their locations have been sensed, the program is already planned to execute the necessary action.

Planning automatic actions that are functions of future measured states is considerably harder than planning with complete prior knowledge. One approach is to produce plans whose primitive elements are guaranteed to work over a whole range of possible execution states [86]. The planning algorithm discussed earlier can be used for the primitive motions. To overcome small part misalignments compliant motion (i.e., motion which follows the surface of an object) may be used, or the tendency of displaced parts to self-align in a gripper may be exploited. The additional concern is to verify that the assumption of robustness is correct, given the maximum allowed variation.

Incomplete planning information introduces a number of significant issues, like representing the geometric variability, the generation of sensor plans, and the generation of parameterized plans [87].

The planner requires a computationally effective means for representing *sets* of object configurations and for predicting the effects of actions over such sets. Typically these sets are described in terms of free scalar parameters corresponding to geometric degrees of freedom (position, shape parameters, etc.) [11,126]. In [30] Donald extended the idea of configuration space by adding another dimension of uncertainty and called it *task parameter.* Figure 1.3 shows an example of a three dimensional configuration space in which one of the dimensions represents the relative position of two blocks and other two dimensions represent the position of the robot hand. The task space, thus formed, is labelled as *generalized configuration*

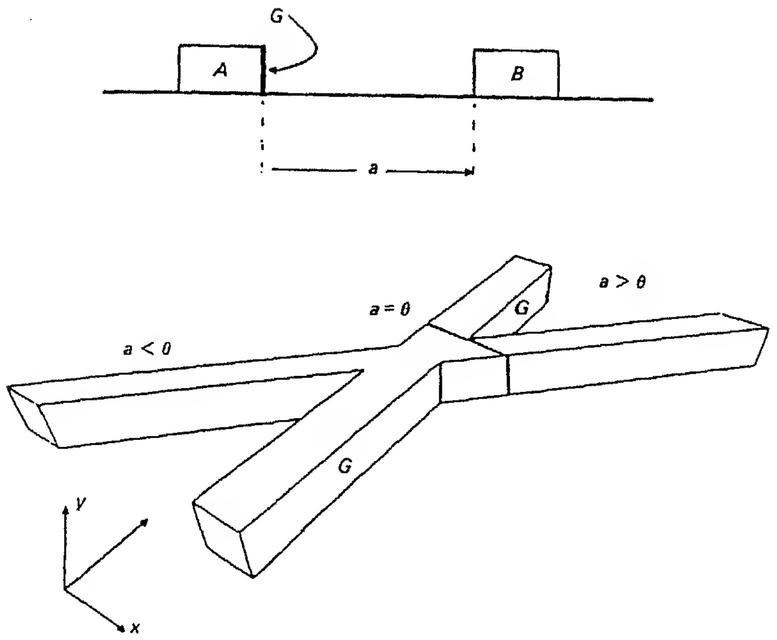


Figure 1.3: Representing variability by adding dimensions in a configuration space [30]

*space* [30]. Thus the generalized configuration space is the original configuration space augmented by as many dimensions as there are uncertain world parameters. This type of modelling enables the planner to treat all the degrees of freedom in the task uniformly, albeit at a large computational cost.

Parameterized plans are essentially subroutines whose inputs are variables representing those elements of the execution time environment that cannot be known at plan generation time [87]. Plans can be thought of as decision trees where the branches have fixed sequence of actions and the nodes test the values of some parameters to choose between alternative sequences. The type of sensing and sensory interpretation depends a lot on the variability of the task. Two important approaches to interpreting sensory data are:

1. Gather data for a relatively small number of features with unambiguous interpretation, e.g., a hole or a corner; each reading generates constraints on

the task parameters. The model of the variability in the parameters can be used on-line to select sensing steps that guarantee unambiguous interpretation [48,110].

2. Gather a large number of less distinctive features, e.g. edge fragments, and use the geometric constraints implicit in the model to prune inconsistent matches [38,47].

### Planning with uncertainty

Due to complete uncertainty, little prior planning can be done. In this situation, the planner must command the appropriate sensing actions and wait until more information is available to do planning.

For most practical applications, the robot would not go *exactly* where it is directed to go, the information reported by sensors is inaccurate and incomplete, the robot's internal model of the world is imprecise and parts do not interact as expected. There are generally two types of discrepancies between idealized models and the actual state of the world. First, there are **errors**, such as an unexpected object along a trajectory or a missing feature on a part, and secondly there is **uncertainty**, such as that arising from the limited precision of the robot, [87] i.e., on relatively small parametric variations between model and reality. A methodology, different from the ones described earlier, needs to be adopted in these circumstances. The following description pertains more to grasping and assembly operations than to motion planning in general.

Geometric variation is often measured through sensing, but the present focus is on *residual uncertainty*, i.e. the remaining uncertainty after all measurement operations that can be planned have been done. The residual uncertainty can be handled by constructing *strategies* that combine compliant motions and additional sensing. These strategies must be planned so as to guarantee that the task can be successfully carried out in spite of the remaining uncertainty in task parameters [87]. The introduction of uncertainty brings forth several important issues: estimating and propagating uncertainty, error detection and recovery (EDR), and planning fine motion strategies.

**Estimating and propagating uncertainty** Expected uncertainty was represented as a distribution [33,117], (typically Gaussian distribution) and statistical measures were computed to obtain narrower estimates of the values of task parameters. The task parameters were represented as random variables with the assigned distribution and such distributions were combined. In [11] expected uncertainty was represented using constraint expressions.

**Error Detection and Recovery** In complex robot applications, more than half of the code is devoted to checking for unusual circumstances [82,126]. Instructions are inserted after every major plan step to verify that that step has proceeded as expected, usually by checking position and force readings against expected values. If an undesirable result is detected, additional instructions attempt to diagnose the problem, correct it, and resume the plan at an appropriate step. In simple cases, it may suffice to repeat the failing step. In extreme cases, the only recourse is to abandon the current task, clear the workspace and begin afresh. The combination of these activities, called *error detection and recovery*, were categorized into the following two situations by Donald [30,31].

1. In some situations, the range of variability of the task is such that the task is impossible. The most common case where this happens is due to the interaction between manufacturing tolerances, e.g., a peg too wide for a hole. These situations are *errors*, and fall under the domain of EDR strategies.

2. In other situations, the goal is unattainable because the environment has some expected surprise in it, e.g., an unpredictable obstacle. This is actually a stronger version of case 1 above, where the environmental variability may include practically anything. Under these circumstances, it is impossible to do any anticipatory planning. The robot will have to rely heavily on on-line sensing in building up its model of the world. Brooks and Chatila have addressed this problem for mobile robots [15,21].

Although an EDR strategy is allowed to fail, it does so recognizably. This means that there is no possibility for the strategy to fail without the controller realizing it. A planner generating EDR strategies in a limited domain has been implemented in [30]. In [32] three approaches to synthesis of multi-step EDR strategies were suggested : these are the *Push-forward Algorithm*, *Failure Mode*

*Analysis* and the *Weak EDR Theory*.

**Fine motion planning** The problem of planning sequences of compliant motions to achieve a goal in the presence of uncertainty but no task variability is called *fine motion planning*. The compliant motion planning requires finding commands that bring one part into a desired relationship with another part such that the executor can detect the termination of each motion. Fine motion planning strategies are described in [86,96,37,30].

A fine motion planning problem is roughly defined as follows. A robot starts out in the *initial region* $I$, a subset of the robot's configuration space. The goal is to reach any point in a specified *goal region* $G$ recognizably bypassing the *obstacle regions* $O_i$ in the configuration space. The term "*recognizably*" means to *reach* the region and to *recognize* that the region has been reached. The objective is to find a sequence of commands that guarantee that the robot will reach *some* point in the goal region from *any* point in the initial region. One difference with gross-motion planning problems is that the start and goal are *regions* instead of single configurations. The new factor in fine motion planning is that compliant motions may be used to carry out tasks where the robot's precision is not sufficient or the uncertainty in the task is too large.

Compliant motion is usually carried out using the *generalized damper* approach [96]. They are used in situations when the robot comes in contact with the environment. It is assumed that the objects are rigid and that the motions are slow enough so that collision forces are negligible. The basic step in planning is the computation of a *set* of configurations for a given commanded velocity vector that are guaranteed to reach the goal region $G$. This new intermediate region is called the *backprojection* of the goal under the commanded velocity. If all the initial robot positions start out within a back projection of the goal, then the goal can be achieved with a single motion. Otherwise, the planner must identify a sequence of subgoals, with the last subgoal being the goal region $G$. The subgoal sequence can be constructed by starting with goal $G$ and repeatedly constructing the backprojections for the previous goal in the sequence. This is called *backward chaining*. The backwards chaining stops when the backprojection includes the starting region $I$.

Another distinct line of approach for dealing with uncertainty was adopted by Lumelsky, Stepanov, Skewis and Sun [89,90,91,92,122,121,93,123,94]. The problem formulation is similar to the central idea presented in this thesis and is briefly described below.

In [93], the authors have proved that this problem is equivalent to developing a strategy for solving a maze. Lumelsky and Stepanov [92] have described several strategies that are guaranteed to find a path if one exists. They have presented two non-heuristic convergent collision avoidance algorithms (Bug-1 and Bug-2) for a point automaton with tactile sensors. In the Bug-2 approach, the automaton begins at the start position, $S$,(see Figure 1.4) and travels to a final target position, $T$. The straight line between these points is the $M$-line. The automaton moves along the $M$-line until it reaches the obstacle boundary, where it defines a hit point, $H_j$. The automaton moves along the obstacle boundary in the local direction until it reaches the $M$-line again, where it defines a leave point, $L_j$, and continues along the $M$-line to the target. Briefly the algorithm works as follows:

*Step (1).* Set $j = 1$.

*Step (2).* Move along $M$-line toward the target $T$ until arrive at $T$ or meet some obstacle; in the latter case, memorize the hit point, $H_j$, and go to Step 3.

*Step (3).* Turn in the local direction and start walking along the obstacle boundary until either arrive at $T$ or meet $M$-line again. In the latter case, if the distance between the current position and $T$ is shorter than that between the hit point and $T$, define the current point as the leave point, $L_j$, update $j = j + 1$, and goto Step 2.

If, however the automaton returns to the last hit point without having defined the next hit point, $H_{j+1}$, then the automaton or the target is trapped. Of course, the path found this way is not likely to be the shortest path possible, although the information gathered as the automaton wanders about, could be used later to find more efficient motions.

Based upon the Bug-2 algorithm, Lumelsky and Skewis [93] then extended this method to develop the Bug-21 and Bug-22 algorithms for a vehicle with a vision radius $r_v$. In the Bug-21 approach, the vision system is used to locate an intermediate target, $T_i$ which is the furthest point within the vision radius along

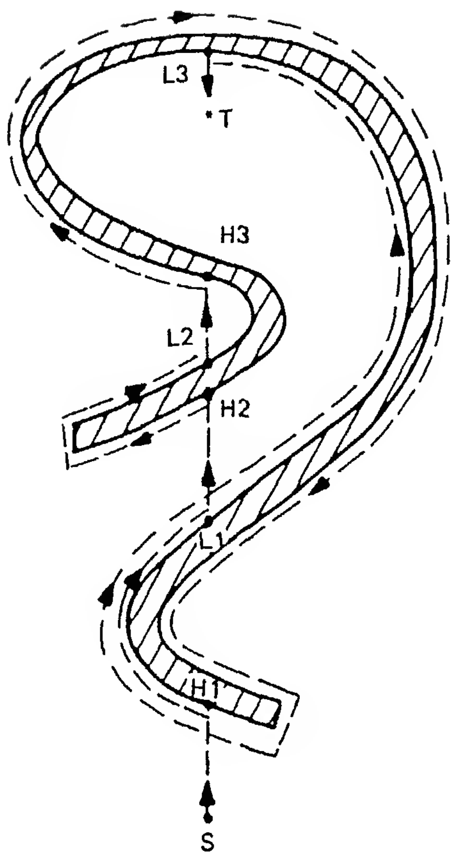


Figure 1.4: Automaton's path (dotted line) under Algorithm Bug-2 [122].

the Bug-2 path. The automaton then heads directly for this intermediate target rather than following the complete Bug-2 path. In most situations this leads to better performance. A practical implementation of the Bug-2 algorithm for an autonomous mobile vehicle has been reported by Kanades et. al [61].

The problem of finding the path of a robot manipulator equipped with sensors amidst unknown obstacles was solved by first topologically transforming the manipulator workspace ($W$-space) to an image space ($I$-space). A feasible collision free path in this manifold was now found using the above strategy. Any physical obstacle in $W$-space was shown to map into a simple closed curve in $I$-space.

Each point in the $I$-space maps back uniquely to its corresponding point in the $W$-space. Thus, a continuous path in the $I$-space will result in a continuous path in the actual workspace of the manipulator. Problems for various kinds of two-link manipulators working on a plane have been solved in [91] and then further extended in [94] to cover 3-dimensional cases.

A special kind of three dimensional cartesian manipulator has been studied in [123] and a non-heuristic algorithm for motion planning has been presented. In this approach the manipulator approaches the target point along the $M$-line and traces the outer boundary of the obstacle when one of the links *touches* an obstacle. The act of *maneuvering around an obstacle* refers to a motion during which the arm is in constant contact with the obstacle. If the arm is simultaneously in contact with more than one obstacle, then effectively all those obstacles will be perceived by the arm as a single obstacle. 3-dimensional curve in $C$-space. Convergence is assured by proving that a finite combination of the path segments is sufficient for reaching the target or for concluding that it cannot be reached. This approach is found to be quite rigorous for non-redundant manipulators.

### 1.2.2 Motion planning specific to navigation problems

The development of many collision avoidance algorithms have been inspired by its potential application in navigation, especially in autonomous mobile vehicles. Navigation is defined as "the science of getting ships, aircraft or spacecraft from place to place, especially the method of determining position, course, and distance travelled" [99]. Although the science of navigation encompasses aircraft, ships and spacecraft, our attention is primarily on autonomous mobile vehicles: The study and development of autonomous mobile vehicles can be broadly classified into [26]:

1. Sensors and Sensing Strategies
2. Position and Course Estimation
3. Map Representation
4. Motion Planning and
5. Kinematics, Control and Trajectory Generation.

It is thus imperative to review the papers pertaining to navigation which have made important contributions to the subject of motion planning.

One of the most crucial problems while dealing with autonomous vehicles working in un-structured environments is the ability to deal with uncertainty. The primary source of uncertainty is known to be at the sensory interface. Erroneous readings from sensors offset the position and course estimation of vehicles. Positional inaccuracy has been shown to be representable through geometric dilution of precision (GDOP) [128] by Torrieri. Torrieri addressed the problem of determining a vehicle's position from simultaneous measurements of sufficient observables to solve the navigation equations uniquely. The position of a vehicle was assessed by observing either naturally occurring landmarks in the environment or through artificially placed beacons. Readings from sensors over a period of time are integrated through the Kalman filtering technique [97,46,98]. Smith, Self and Cheeseman have described a method to manipulate navigation frames optimally [118]. It addresses such questions as whether a mobile robot is expected to have a particular reference object within its field of view. The Kalman filter is used and it is shown how the uncertainty of a navigation frame can be reduced by additional sensing. Cox [25] has described a navigation strategy which works with a prior map of the environment. Range data points obtained from infra-red sensors are matched to the map using a matching algorithm that solves the correspondence problem (for small disparity) by associating data points with their closest Euclidean line segment. A least-mean-square optimization procedure is then applied to determine the vehicle's position. Ayache [2] describes a navigation based on the dead reckoning system whereby the estimate of a vehicle's position is refined by taking successive reading from different locations. The system also constructs a map of its environment as it navigates.

The construction of an appropriate map suitable for navigation is an important topic. Elfes [34] suggests a hierarchy of maps depending on the needs of the computation involved. In deciding on the level of detail to include in a map, there is often a trade-off between offering sufficient detail for the vehicle to perform its duties and too much detail, which makes computation infeasible. Multiple hierarchies based on (1) abstraction, (2) geography, and (3) resolution is a possible solution to this problem. The basic representation used by Elfes (for navigation of autonomous vehicles) is a two-dimensional grid of cells. Each cell contains the occupancy status (unknown, empty, or occupied) along with an uncertainty factor. The information needed to construct this grid is obtained by overlapping sonar readings as the vehicle traverses the region. Ikegami et al. [59] have proposed a similar representation in which the grid cells contain the distance to the closest border. Another method for the representation of occupancy grid information is a quadtree (or octree in three dimensions). A numerical representation of uncertain and incomplete sensor knowledge through *certainty grids* has been proposed in [22,100]. Readings from sensors including sonar, stereo vision, proximity and contact sensors incrementally maintained a probabilistic geometric map of the mobile robot's surroundings, through the certainty grid representation. Another interesting approach for three dimensional representation of the robot workspace for a mobile vehicle is reported in [119]. A cube-based 3-D model was proposed which used a combination of matrix and octree structures for internal data representation. Algorithms for cube mapping, for a cartographer and for route finding were also described.

Navigation systems based on ultrasonic sensors are quite popular and have been researched extensively [27,28,7,8,9,10]. Borenstein and Koren [8] introduced the concept of Virtual Force Field (VFF) for fast obstacle avoidance of mobile robots. This concept was later modified to the Vector Force Histogram (VFH) [9] which permitted the detection of unknown obstacles while *simultaneously* steering the mobile robot toward the target. The VFH method uses a 2-D cartesian histogram grid as a world model. This world model is updated continuously with range data sampled by on-board range sensors. A further improvement of this concept in the form of Histogramic In-Motion Mapping (HIMM) was reported recently [10] for real-time map building.

The problem of coordinated motion of bodies on a plane has received attention recently [6,78]. In [6] a navigation strategy in the presence of moving obstacles on the basis of visual information was proposed. Optimizing strategies for planning under uncertainty using local information based on sensory data was also suggested. In [78] the collision-free coordinated motion of multiple moving robots was solved using a Petri net. The motion constraints of the robot in their paths was arranged as its firing rules and collision-free coordination was planned by manipulation of the firing rules.

The concept of potential functions has been improvised for solving navigation problems with complete prior knowledge leading to "good" definitions of potential functions on manifolds with boundary [67]. Rimon and Koditschek [107] have defined the notion of a *navigation function* which is a refinement of the potential function having the condition of a single minimum (at the desired destination), a uniform finite height at all the boundaries along with several other technical requirements. An extension of the potential function approach in domains requiring contact with the environment is reported by Hogan [56]. It demonstrates that the potential function approach to unified task description and control is not limited to task domains involving a purely geometric environment. A good review of the penalty function methods appears in [68].

Planning algorithms that generate paths in task-space to be directly followed by vehicles, must consider a few additional constraints since robot vehicles are non-holonomic systems and hence not capable of arbitrarily rotating and translating [68]. All feasible paths in the vehicles free-space might not be practical to achieve. Laumond [76] has shown that if two configurations are connected by a path in configuration space that does not correspond to a feasible motion, then there exists another path between the configurations that does correspond to a feasible motion. Therefore the problem of determining if a feasible motion of a nonholonomic vehicle exists or not, is computationally equivalent to determining the path connectivity of the free space. However, the problem of actually computing a feasible motion between two configurations that are known to be path connected in free space, has no known complexity bounds. In addition to the fact that many robot vehicles are non-holonomic systems, many also have a lower bound on their turning radius, due to dynamic constraints, for example. The path that such a vehicle can follow has a bound on its curvature. Laumond [76] has discussed the problem of computing a bounded curvature path in the presence of obstacles when reversals are allowed. The general problem of path planning under the bounded turning radius constraint is difficult when reversals are not allowed and so compromises are necessary in order to find efficient algorithms. A path planning method for a mobile robot subject to non-holonomic constraints by defining a set of canonical trajectories which satisfy them, was described by Jacob and Canny [60]. The control of mobile vehicles equipped with sensors, in a dynamic environment, has been reported by Griswold and Eem [49] where an optimization approach is adopted which provides for the acceleration and deceleration of the vehicle and also the time for which this control must be applied to avoid a collision. The constraints on the objective function are given in terms of collision probabilities. The emphasis is put on controlling the motion along the assigned path rather than changing the path itself. Zhu [139] proposed a model and some control strategies for dynamic obstacle avoidance in visual navigation of mobile robots based on a hidden Markov model. The aim was to find a collision-free trajectory that took into account any possible motion of obstacles in the local environment, was consistent with a global goal or plan of the motion, and moved the robot at as high a speed as possible subject to kinematic constraints.

### Use of neural networks for navigation

The development of neural networks has unleashed a powerful tool for solving complex systems. The collision avoidance for multiple moving bodies has also be attempted by the technique of neural optimization to work on real-time, as reported by Lee and Bein [77]. The positions or configurations of robots were taken as the variables of the neural circuit, and the energy of network was determined by combining various functions in which one function was chosen to make the robot approach to its goal and another to prevent each robot from colliding with other robots or obstacles. This approach was valid only for a completely known environment.

A learning neural network for collision between autonomous mobile vehicles was proposed by Tuijnman and Kröse [129]. The basic idea was that the network should learn from example encounters between robots that are properly handled. An example was presented with two moving automatons moving on a plane.

### 1.2.3 Application of fuzzy logic for robot control

Fuzzy control is a well developed field which has been proved to be useful in a wide variety of applications [75,95]. Scharf et. al. [108] have described a rule-based self-organizing algorithm for the control of a robot arm. The algorithm requires only minimal knowledge of the often highly non-linear robot dynamics. Step response and tracking tests have also beenreported on a real robot. The fuzzy rules are formulated in the form IF (Error is $X$ AND Change-in-Error is $Y$) THEN (Output should be reinforced by $Z$). Ciliz and Isik [23] have described a rule-based motion controller for an autonomous mobile robot using sensory data. The assumed inexactness in world description is represented by fuzzy memberships, and the state space is discretized into a linguistic vocabulary. Fuzzy motion control rules have been experimentally derived, and used in a fuzzy inference mechanism to give the final control command to robot actuators.

### 1.2.4 Application of fuzzy logic for sensor data interpretation

#### Use of sensors in robotics

Sensors are an integral part of any robotic application because they form a bridge between the robot's internal state and the external world. They are vital for interacting with the environment in a flexible manner which is necessary for working intelligently in unstructured environments.

The function of robot sensors may be divided into two principal categories [44]: *internal state* and *external state*. Internal state sensors deal with the detection of variables such as arm joint position, which are used for control. External state sensors deal with the detection of variables such as range, proximity and touch. External sensing is used for guidance as well as for object identification and handling.

External state sensors may be further classified as *contact* and *noncontact* sensors. As their name implies, the former class of sensors respond to physical contact, such as touch, slip, and torque. Noncontact sensors rely on the response of a detector to variations in acoustic or electromagnetic radiation. The most prominent examples of noncontact sensors measure range, proximity, and visual properties of an object. The primary concern of this thesis is in range sensing which is further elaborated in Section 3.2.

### Modelling sensors through fuzzy parameters

The design of an intelligent robot controller is commonly achieved by use of artificial intelligence techniques and expert systems. Use of such methods requires representation of knowledge qualitatively, i.e., at a higher level through symbolic notation. However, a limitation is that the controller loses it ability to deal with low level data processing, i.e, at the numeric level [43]. An approach which represents data both at the symbolic and numeric level is suggested in [43]. The symbolic level representation is achieved by means of a fuzzy representation of numeric values lying between a known maximum value. The fuzzification is done on the basis of two fuzzy concepts $C_1$ and $C_2$ which are defined simply as

$$\begin{aligned} \mu_L(C_1, y) &= (100 - y)/100 \qquad\qquad \text{with } 0 \leq y \leq 100 \\ \mu_L(C_2, y) &= y/100 \end{aligned} \tag{1.2}$$

The other quantities are found by applying modifiers (see Chapter 2 for more details) to these two basic quantities.

In [42] a fuzzy logic based "hybrid" system has been developed in which different models for handling vagueness are coupled together to obtain a more flexible knowledge representation paradigm for interpreting sensory data. The activity of a sensing is represented by a layered schema: the sensor level, data fusion level and the reasoning level. The sensor level is the numeric level, the data fusion level is the symbolic level while at the reasoning level, decision and planning activities are done. An application of fuzzy logic for sensor data interpretation was reported by Sood et.al. [120] for interpreting the forces and torques generated by a force/torque sensor during an assembly operation involving insertion of printed circuit boards. The use of fuzzy logic for coping with noise was reported in [16] where an approach for analyzing motion sequences from a mobile robot operating in a dynamic environment was presented. The usual concept of Focus of Expansion (FOE) was extended to a Fuzzy FOE which defined an image *region* rather than a single point.

## 1.3 Present Trends Versus Proposed Work

An important area of research towards the development of autonomous systems is the ability to deal with uncertainty. Given that uncertainty is an unavoidable aspect of any navigation problem, specially in outdoor environments, it is necessary to develop methods that can ensure that a task can be achieved in spite of the uncertainty [84]. The decision making capacity of a mobile robot in the face of uncertainty is inherently linked to its sensing ability. But using more accurate sensors is not expected to achieve the task alone. The *actions* of the robot must be chosen by exploiting sensed data. 'The traditional way of encoding the action plans of the robot as a fixed sequence of actions is bound to fail here. Instead, the robot's action plans must be encoded as a function of the state of the world. In this sense, a robot plan is like a feedback loop: a mapping from sensors to actuators. But unlike a feedback loop, a robot plan should be a function from states of the world to goals for the robot, not robot actions directly. The goals can then be mapped into actions by geometric planners that can take into account the current state of the world and the capabilities of the robot.' [84]

The current thrust for the development of task-oriented languages requires the robot controller to interpret high-level language commands while incorporating sensor information. The robot-level commands are to be generated automatically from such descriptions [85]. In unknown environments, the importance of coupling sensory information with robot commands assumes special significance. The major hurdle is in using sensor information to update the knowledge of the robot's internal states, the external world and the action of the robot. The requirement is thus to codify the "sensor knowledge" and then to integrate with the "prior knowledge" of tasks and actions, so that the system can detect all the supposed faults and attempt to recover from them producing a new action plan. Representation of *uncertainty* becomes one of the primary issues to be addressed. Therefore, a paradigm which employs a *qualitative* representation of uncertainty rather than a *quantitative* one is needed.

A primary source of uncertainty is at the sensory interface. Sensors normally give a numerical evaluation of some physical quantity by means of direct or indirect measurements. The reliability of such sensory data may be affected by *systematic errors* or *random errors* [42], the former due to the particular way in which the experiment is managed, and the latter induced by noise. 'Moreover, errors can propagate because of the quantization by the digital equipment. Methods to reduce the sources of this imprecision exist, but they are too costly to be used in robotic applications. For example, a proper experimental setup may eliminate systematic errors, while statistical methods may be employed to deal with noise.' [42] These approaches require additional hardware and software bringing in extra complications for the whole system.

Ferrari and Chemello [42] floated an idea to handle the intrinsic uncertainties that exist in the process of sensing. According to them, 'it is worthwhile to note that humans have a good ability in managing incomplete data in a qualitative way, extracting significant information from the real world and making decisions on the basis of very rough representations.' Similarly, designers of intelligent autonomous systems cannot improve their performance by merely using the most sophisticated hardware. Instead, a proper *knowledge representation* is needed to model uncertain situations and incomplete information. As a consequence, the numeric data coming from sensors should be integrated with the symbolic representation of tasks and actions. This can be achieved by establishing a link between the numeric sensory value with its symbolic representation through a primitive logical concept.

Strategies for avoiding collision between autonomous mobile vehicles could be based on local decision making in which each vehicle makes a decision on the basis of its own observations, assuming that there is ample space to maneuver. An analogous situation occurs while avoiding collision between two ships at sea [129]. Each ship has to act according to certain rules (International Regulations for Preventing Collision at Sea) [24] and has no knowledge about the other ships apart from the observations that have been made.

This approach may also be chosen for collision avoidance between autonomous vehicles because of the following reasons [129]:

1. It seems to be quite effective in avoiding collisions at sea, since ships usually do not collide.

2. It can also be used for avoiding collisions with human controlled vehicles.

3. This approach would be useful as a safety mechanism in combination with another approach such as a central decision making system or global path-planning.

The existence of a set of rules might suggest that the collision avoidance problem is solved, and it merely needs to be appropriately encoded to a computer representation. This is however deceptive, since the rules are vaguely worded, and though they do prescribe certain actions, they state neither the timing nor the magnitude. Furthermore, they are limited to encounters between only two ships and the prescribed action must be carried out only if 'the circumstances of the case admit' [129] which is ambiguous.

In view of the facts mentioned in the above paragraphs, we make the following observations:

1. There is a need for motion planning strategies that can work efficiently under uncertainty. Till now, representation of uncertainty still remains one of the biggest issues. Since these systems must work on-line, speed of decision making is also an important factor.

2. It is necessary to decouple robot goals from its action plans. The sensory readings must be used to come up with goals of the robot, which must later be converted into movements of actuators.

3. The subject of qualitative description of motion planning strategies has not received much attention, though it seems to be the natural way most bio-systems work. A qualitative description will not require much accuracy of world description, which in turn reduces the burden of building an accurate description from sensory readings.

4. There exists a need for integrating sensory information with robot commands so that it can be used effectively to update the system's knowledge about the external world and its action. Thus it is necessary to evolve a strategy for working with codified sensor knowledge for decision making.

Fuzzy set theory offers a powerful way of associating symbolic representations to numeric values. It is an easy-to-use theoretical tool to represent intrinsic uncertainty and ambiguity together into a set called a *Fuzzy Set* and has been used in [42,43]. This work proposes to extend the fuzzy set theoretic approach to the motion planning problem as it is expected to provide some advantages over the present approaches. A fuzzy set allows its elements to be given a *grade of membership* which encodes the extent to which that element belongs to the set. A fuzzy set can thus be associated with each sensor (or group of sensors), to obtain a fuzzy measurement of some physical quantity. The assignment of a meaning to a fuzzy set through a membership function is not unique, but it is to be noted that this arbitrariness is inconsequential, because the major emphasis is on the *shape* of the membership curve rather than on the punctual values.

Use of fuzzy logic leads to flexibility and computational efficiency. A rule paradigm lets people easily try out different strategies, add new heuristics and modify old ones with minor changes to the whole system. The complexity of the program implementation is concealed inside the fuzzy rules, and provides an easy interface to the user to fine-tune the algorithm to suit different requirements. When a new sensor is introduced, it is sufficient to give its correct definition in terms of an appropriate fuzzy set.

It is to be noted that, till date, there is no automated system that outperforms or even matches the dexterity of the collision avoidance maneuvers practiced by a human driver or ship pilot. Thus, it gives us the incentive to investigate the possibility of encoding 'experience' through a fuzzy controller, that commands a move based on instantaneous sensory information, akin to one commonly practiced by us. The *avoidance maneuvers* can be expressed through fuzzy rules that can be modified to change the 'expertise' of the system. It is also possible to incorporate a learning controller, in which the system is left to work in a simulated environment whereby it modifies its own rule set to tune itself to the desired situation. In this way the system can be made to finally emulate the experience of a human operator in a limited domain and finally be put to work on a real site.

## 1.4 Objective and Scope of Present Work

This work attempts to solve the collision avoidance problem in a dynamic environment through an approach involving fuzzy logic. A point-wise list of the major objectives is presented in the following paragraphs.

1. *An attempt has been made to use the concepts of Fuzzy Set Theory for a qualitative description of the world model.*

A qualitative description of sensory data helps to ignore minor inaccuracies and bridges the gap between low-level numeric data and a higher level sensory knowledge representation. A qualitative description of sensory knowledge is needed for higher level decision making. A further advantage of this representation is the lower demand it places on sensor accuracy.

2. *A general approach to a fuzzy rule based collision avoidance strategy is attempted. This rule set is based on the qualitative description of the world.*

For most practical motion planning problems too much accuracy is not required. Such situations occur in navigation and in manipulation within relatively uncluttered workspaces. Thus an attempt has been made in this thesis to develop an algorithm which does not require an accurate world description. Consequently the motion planning is also done in an approximate manner. This is achieved by employing more physical terms like "near", "too near", "far" etc.. The avoidance measures are stated in the form of *fuzzy rules* which encodes the expertise of the system. Some advantages expected by this approach are flexibility, simplicity, speed and robustness at the expense of accuracy.

3. *The fuzzy logic based methodology is proposed to solve navigation problems through simulation runs where collision is to be avoided between multiple bodies moving in a dynamic environment based on instantaneous description of the world model.*

A common problem of navigating a body amidst other moving bodies is attempted. It is assumed that no prior information about the environment is available and avoidance measures are to be adopted on the basis of instantaneously available local sensory data. Some kinematic constraints on the body are imposed and the resulting collision avoidance maneuvers are observed.

4. *A further attempt has been made to solve the collision avoidance problem for manipulators in a dynamic environment, using the same fuzzy logic approach but with modified fuzzy rule set. The effectiveness of the proposed algorithm is assessed through simulation runs.*

    1. *First a planar and redundant manipulator with rotary joints is considered.*
    2. *Then the proposed algorithm is tried for PUMA-560 where the desired redundancy to avoid collision has been introduced by foregoing some orientational parameters.*

Intelligent avoidance measures are to be incorporated by appropriately framing the fuzzy rules while maintaining the kinematic constraints of the manipulator.

5. *It has been endeavoured to write the computer program in an object-oriented language (namely C++), with the idea of having an extensible program, so that it is easy for future modifications and for interfacing with real manipulators or vehicles.*

Hardware interfacing in future is expected to be easy because it only requires replacing the appropriate software 'objects' with the associated hardware element.

## 1.5 Overview of Thesis

This thesis is divided into seven chapters. Chapter 2 reviews the relevant mathematics for building a fuzzy controller. The methodology for casting a motion planning problem to a fuzzy control problem is introduced in Chapter 3. This notion is applied to three different problems in the following three chapters. The problem of collision avoidance of moving bodies on a plane are discussed in Chapter 4. The concepts developed in this chapter may find application in navigation.

Besides demonstrating the use of collision avoidance strategies for multiple moving bodies using only instantaneous information, this chapter also tries to investigate the effect of introducing purely geometric constraints on the motion of the moving bodies. Based on the restrictions put on the path two kinds of bodies are defined — those that can deviate from the assigned path and those that cannot. In Chapter 5 the same logic is applied to a planar rotary manipulator for collision avoidance and a separate algorithm for circumventing an obstacle is developed. A provision has been kept for completely sacrificing the task of the manipulator when danger levels become menacingly high. In this condition, the manipulator *drifts* along with the moving obstacles to avoid a collision. Although this process leads to unproductive work, it is conceived as an emergency measure to fulfill the primary objective of collision avoidance when the situation demands it. Chapter 6 extends the concepts of Chapter 5 to three dimensions and demonstrates the application of a simplified version of this algorithm on a real manipulator through simulation runs. It is shown that the PUMA-560 can avoid a collision with both static and dynamic obstacles in the workspace using sensors mounted on its links. This chapter also gives the details of implementing the algorithm in an object-oriented language. Finally Chapter 7 ends with some observations and suggests scope for future work.

*So 'ere's to you, Fuzzy-Wuzzy,*
*at your 'ome in the Soudan;*
*You're a pore benighted 'eaten*
*but a first-class fightin' man.*
[Fuzzy-Wuzzy]
– RUDYARD KIPLING

# Chapter 2

# Theory of Fuzzy Sets and Fuzzy Controller : A Review

## 2.1 Introduction

Fuzzy Logic was evolved as a natural outcome of the inability of classical mathematics to describe imprecise situations. A mathematical description is very precise and is used to describe models of empirical phenomenon in the "hard sciences" such as engineering, chemistry or physics. A common disadvantage is that the excessive complexity of reality forces us to simplify the model as much as possible, and hence the mathematical description often does not work. Moreover, some aspects of the real world always escape such precise mathematical models. To meet the need of developing a mathematics which can give a unifying view to the notion of inexactness, the theory of fuzzy sets was introduced by Zadeh [135]. In that paper, Zadeh called for "a mathematics of fuzzy or cloudy quantities which are not describable in terms of probability distributions".

The main idea of this theory is very simple and natural. Since we are not able to describe the belongingness by a vague notion, let us replace this decision by a *measure* from some scale expressing its role. If this measure is high, the element belongs to the class with a higher degree of certainty. This measure will be called "grade of membership" in the given class, and the class in which each element is characterised by its membership grade, will be called the fuzzy set. The grade of membership can also be viewed as a degree of our certainty (or belief) that the element belongs to the given fuzzy set.

The theory of fuzzy sets has as one of its aims the development of a methodology for the formulation and solution of problems that are too complex or too ill-defined to be susceptible of analysis by conventional techniques. Robot motion planning is a complex problem that, though has been solved completely theoretically, still lacks the efficiency to be solved in real-time, even on high-speed computers. Moreover, it happens to be one of the most common problems that humans seem solve quite effortlessly in daily routine. A rigourous solution for the motion planning problem holds good when the environment is completely known apriori, and thus an off-line algorithm offers a neat solution. But when the environment is not known apriori, and has to be sensed on-line (through external sensors), resort must be taken to an algorithm that is capable of taking decisions on-line. It is our attempt , thus, to corelate some facts from fuzzy set theory and motion planning, to evolve a simple control strategy, that, though not very exact, will yield a reasonable solution in real-time. The present chapter reviews the basic concepts of fuzzy logic which are then applied to motion planning problems in the following chapters.

## 2.2 Set Theory

### 2.2.1 Basic definitions and operators

The fundamental basis in mathematics that helps to construct all mathematical objects is the concept of set theory. The primary notions of set theory are those of "class" and "membership". The object $Y$ belongs to the class $X$ iff it has the property $\phi$. The class $X$ is determined by the property $\phi$, thus

$$X = \{Y, \phi(Y)\} \tag{2.1}$$

If $Y$ is an element of $X$, then we write $Y \in X$ and in the opposite case, $Y \notin X$ (or $\neg(Y \in X)$ where ' $\neg$ ' denotes the negation). The class $X$ itself can be an element of another class $Z$, $X \in Z$. Such a class $Z$ is called a "set".

Basic Operations :

A) Two classes are *equal* iff they have the same elements, i.e. if $x \in X \Rightarrow x \in Y$ and $x \in Y \Rightarrow x \in X$. A class $X$ is a *subclass* of a class $Y$, $X \subseteq Y$, if $X \in X \Rightarrow x \in Y$. Hence, $X = Y$ iff $X \subseteq Y$ and $Y \subseteq X$.

B) The *intersection* of classes X, Y is a class $X \cap Y$ such that $x \in X \cap Y$ iff $x \in X$ and $x \in Y$ holds for every $x$. The empty class $\phi$ is

$$\phi = \{x; x \neq x\}\,.$$

Classes X, Y for which $X \cap Y = \phi$ holds, are called *disjoint classes.*

C) The union of classes X, Y is a class $X \cup Y$ such that $x \in X \cup Y$ iff $x \in X or x \in Y$ holds for every x. The class containing all the sets is called the *universal set* and is defined by the formula

$$V = \{x; x = x\}\,.$$

D) Let $x$,$y$ be two sets and let

$$\langle x, y\rangle = \{\{x\}, \{x, y\}\}.$$

$\langle x, y\rangle$ is called the *ordered pair* of $x$ and $y$. Then $\langle x, y\rangle = \langle u, v\rangle$ iff $x = u$ and $y = v$.

The *cartesian product* , (symbolically denoted as $\times$ ) of two classes $X$ , $Y$ is a class

$$X \times Y = \{\langle x, y\rangle; x \in X \text{ and } y \in Y\}$$

i.e. the class of all the ordered pairs whose first element belongs to $X$ and the second element belongs to $Y$.

### 2.2.2 Relations and Structures

#### Relations

A) A *binary relation* is a subclass $R$ of a Cartesian product i.e.

$$R \subseteq X \times Y.$$

If $\langle x, y\rangle \in R$ then $x$ and $y$ are in the relation $R$ and we sometimes write $xRy$. Given two relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, the composition relation is given by

$$R \circ S = \{\langle x,z\rangle; \exists y \text{ such that } \langle x,y\rangle \in R \wedge \langle y,z\rangle \in S\}$$

where $R \circ S \subseteq X \times Z$.

B) A binary operation $\otimes$ on the class $X$ is a function

$$\otimes : X \times X \mapsto X.$$

We will write $z = x \otimes y$ instead of $\langle x, y, z\rangle \in \otimes$ where $z$ is the result of the operation $x \otimes y$.

An *ordering* is a binary relation $\preceq$ which is

- reflexive, i.e. $x \preceq x$
- antisymmetric, i.e. $x \preceq y$ and $y \preceq x$ implies $x = y$
- transitive, i.e. $x \preceq y$ and $y \preceq z$ implies $x \preceq z$

holds true for every $x, y, z \in A$.

Let $B \subseteq A$. Then an *upper bound* of $B$ is an element $m \in A$ such that $y \preceq m$ holds for every $y \in B$. Analogously, a *lower bound* of $B$ is an element $n \in A$ such that $n \preceq y$ for every $y \in B$. The *least upper bound* is called the *supremum*

$$\sup_A(B) = \bigvee_{x \in B} x \tag{2.2}$$

and the greatest lower bound is called the *infimum*

$$\inf_A(B) = \bigwedge_{x \in B} x \tag{2.3}$$

## Structures

A *structure* is a set together with a collection of operations and relations (in general $n$-ary).

A *lattice* is a structure which satisfies the axioms of associativity, commutativity and absorption. For example, let us consider the lattice $\langle A, \vee, \wedge\rangle$ where $\vee$, $\wedge$ are called *join* (supremum) and *meet* (infinimum) respectively. Then for $x, y \in A$ the absorption property is $(x \vee y) \wedge x = x$ and $(x \wedge y) \vee x = x$ hold true for every $x, y \in A$.

We can define an ordering on any lattice using one of the two equivalent conditions :

$$x \preceq y \text{ iff } x \wedge y = x$$

$$x \preceq y \text{ iff } x \vee y = y.$$

The operations $\vee$ , $\wedge$ correspond respectively to the operations of supremum and infimum defined earlier. Hence,

$$\sup_A(\{x, y\}) = x \vee y$$

$$\inf_A(\{x, y\}) = x \wedge y$$

## 2.3 Fuzzy Sets

### 2.3.1 Important definitions

**Definition** : Let U be a set (which is also called universe). Let $\mathcal{L} = \langle L, \vee, \wedge, \mathbf{1}, \mathbf{0} \rangle$ be a lattice. The *fuzzy set* $A$ in the universe $U$ is a function expressed as

$$A : U \mapsto L.$$

$\square$

The function $A$ is usually called the *membership function* of the fuzzy set $A$. For each element $X \in U$, there is a corresponding element $Ax \in L$. $Ax$ is called the *membership grade* of $x$ in the fuzzy set A. If $Ax = 0$ then $x$ does not belong to $A$. If $Ax = 1$, then $x$ belongs to $A$. If $Ax \neq 0, 1$ then $x$ partly belongs to the fuzzy set $A$.

If $A$ is a fuzzy set in universe U, then we will write

$$A \sqsubseteq U.$$

It means that we can associate a grade of membership $Ax$ for each element $x \in U$. If the membership function $A$ has only two values, namely 1, 0; then A becomes the characteristic function of an ordinary set and, hence A is a set in the conventional sense which we will henceforth refer to as crisp set.

The definitions below are followed by an example for clarification.

**Definition**

a) The *support* of fuzzy set $A$, denoted by $Supp(A)$ is a classical set

$$Supp(A) = \{x; Ax \neq 0\}.$$

b) The *$\alpha$-cut* of the fuzzy set $A$, for the given $\alpha \in L$ , is a classical set

$$A_\alpha = \{x; Ax \wedge \alpha = \alpha\} = \{x; \alpha \leq Ax\}.$$

c) The *$\alpha$-level* of the fuzzy set $A$ for a given $\alpha \in L$ is a classical set

$$A^\alpha = \{x; Ax = \alpha\}.$$

d) The *kernel* of the fuzzy set $A$ is a classical set

$$Ker(A) = \{x; Ax = 1\}.$$

e) The *height* of a fuzzy set $A$ is is defined as

$$Hgt(A) = \bigvee \{Ax; x \in U\}.$$

If $Ker(A) \neq \phi$ then $Hgt(A) = 1$. The converse is not true. The fuzzy set $A$ is *normal* if $Ker(A) \neq \phi$. It may sometimes be referred to as a set whose height $Hgt(A) = 1$.

f) The cardinality of a fuzzy set can be defined as follows. Let $A : U \mapsto \langle 0, 1 \rangle$ and Supp(A) be a finite set. Then the cardinality of the fuzzy set $A$ is the number

$$Card(A) = \sum_{x \in Supp(A)} Ax.$$

□

We will use the following notation to explicitly write down the fuzzy set.

$$A = \bigcup_{x \in U} \frac{Ax}{x} \tag{2.4}$$

where the symbol $\frac{Ax}{x}$ denotes an ordered pair $\langle Ax, x \rangle \in L \times U$. If $Supp(A)$ is finite, i.e. $Supp(A) = \{x_1, x_2, \ldots, x_n\}$ then (2.4) is also represented as

$$A = \left\{ \frac{Ax_1}{x_1}, \frac{Ax_2}{x_2}, \ldots, \frac{Ax_n}{x_n} \right\} \tag{2.5}$$

where $Ax_i \neq 0$ for each $i = 1, 2, \ldots, n$. We will use the latter representation for representing a fuzzy set in this work.

*Example* : Let $L = \langle \mathbf{0}, \mathbf{1} \rangle$ where $\vee = \max$ and $\wedge = \min$, $\mathbf{0} = 0$, $\mathbf{1} = 1$ and let $U = \mathsf{N}$ (set of natural numbers). Let us take a fuzzy set as

$$A = \left\{ \frac{0.1}{1}, \frac{0.6}{2}, \frac{0.5}{4}, \frac{0.7}{6}, \frac{1}{9}, \frac{1}{10} \right\}$$

in which $x_1 = 1$, $x_2 = 2$, $x_3 = 4$ and so on, and $Ax_1 = 0.1$, $Ax_2 = 0.6$ , $Ax_3 = 0.5$ and so on.

It is easy to observe that

$$\begin{aligned} Supp(A) &= \{1,2,4,6,9,10\} \\ Ker(A) &= \{9,10\} \\ A^{0.3} &= \{2,4,6,9,10\} \\ A^{0.5} &= \{4\} \end{aligned}$$

This fuzzy set is normal. □

Let $\mathrm{L} = \langle L, \vee, \wedge, \mathbf{1}, \mathbf{0} \rangle$ be a lattice. This lattice L will be called a residuated lattice if we can define two binary relations $\otimes$, $\rightarrow$ (called *multiplication* and *residuum*, respectively) fulfilling the following conditions :

**A**) The operation $\otimes$ satisfies the property

$$\delta \otimes \alpha \leq \delta \otimes \beta$$

for every $\delta \in L$ and for $\alpha, \beta \in L$ with $\alpha \leq \beta$. The above property is called *isotone* property.

B) For every $\delta \in L$ and for $\alpha, \beta \in L$ with $\alpha \leq \beta$, the operation $\rightarrow$ is *antitone* in the first variable $\alpha$ and isotone in the second variable $\beta$. i.e.

$$\begin{aligned} \alpha \rightarrow \delta &\geq \beta \rightarrow \delta && \text{antitonicity} \\ \delta \rightarrow \alpha &\geq \delta \rightarrow \beta && \text{isotonicity} \end{aligned}$$

hold for every $\delta \in L$ and $\alpha, \beta \in L$ such that $\alpha \leq \beta$.

c) The above two binary operations satisfy the property of adjunction, i.e.

$$\alpha \otimes \beta \leq \gamma \text{ iff } \alpha \leq \beta \rightarrow \gamma$$

holds for every $\alpha, \beta, \gamma \in L$. The couple $\langle \otimes, \rightarrow \rangle$ is called the adjoint couple of operations.

*Example :* Let $L = \langle 0, 1 \rangle$ and put $\vee = \max$, $\wedge = \min$, $\mathbf{1} = 1$ , $\mathbf{0} = 0$. Then we can define an adjoint couple $\langle \otimes, \rightarrow \rangle$ such that

$$\begin{aligned} \alpha \otimes \beta &= 0 \vee (\alpha + \beta + 1) \\ \alpha \rightarrow \beta &= 1 \wedge (1 - \alpha + \beta) \end{aligned} \tag{2.6}$$

for $\alpha, \beta \in \langle 0, 1 \rangle$ and '+' and '-' are the operations of addition and subtraction of real numbers. It can be verified that the operation $\otimes$ and $\rightarrow$ satisfy the properties defined earlier.

The operation $\otimes$ as defined in (2.6) has the additional property that

$$\alpha^n = \alpha \otimes \ldots \otimes \alpha = 0$$

for any $n \geq \frac{1}{1-\alpha}$. $\alpha^n$ is often called the *bold product.* □

The operation residuum $\rightarrow$ is used to define another operation called *biresiduum* as follows

$$\alpha \leftrightarrow \beta = (\alpha \rightarrow) \wedge (\beta \rightarrow \alpha).$$

Biresiduum serves as a natural interpretation of equivalence in fuzzy logic.

### 2.3.2 Operations on Fuzzy Sets

**Definition** : Let $A \sqsubseteq U$, $B \sqsubseteq U$ be fuzzy sets and $\mathrm{L} = \langle L, \vee, \wedge, \otimes, \rightarrow, \mathbf{1}, \mathbf{0} \rangle$ be a residuated lattice. The basic operations on fuzzy sets are defined as

| | | |
|---|---|---|
| *Union* | : | $C = A \cup B$ iff $Cx = Ax \vee Bx$ |
| *Intersection* | : | $C = A \cap B$ iff $Cx = Ax \wedge Bx$ |
| *Bold Intersection* | : | $C = A \bowtie B$ iff $Cx = Ax \otimes Bx$ |
| *Complement* | : | $C = \overline{A}$ iff $Cx = Ax \rightarrow \mathbf{0}$ |

for every $x \in U$. □

Further, if $L = \langle 0,1 \rangle$ and $\otimes$ and $\rightarrow$ are as defined in (2.6) then,

$$\begin{aligned}(A \bowtie B)x &= 0 \vee (Ax + Bx - 1) \\ \overline{A}x &= 1 - Ax\end{aligned}$$

hold for every $x \in U$.

**Residuum on fuzzy sets** : This residuum is denoted by $\ominus$ and is defined as follows. For every $x \in U$ and for two arbitrary fuzzy sets $A$ and $B \sqsubseteq U$,

$$C = A \ominus B \text{ iff } Cx = Ax \rightarrow Bx.$$

$C \sqsubseteq U$ is called the residuum of $A$ and $B$.

### 2.3.3 Fuzzy Relations

The fuzzy relations are useful to express cases when it is not possible to distinguish clearly the relations between two objects. For example, the commonly used relation 'much smaller than', is a fuzzy relation because in the expression $5 \ll 1000$ we do not quantify as to "how much smaller" 5 is compared to 1000. Thus $5 \ll 50$ and $5 \ll 1000$ both are true, but the term "much smaller" has different strength in the two cases. More precisely, we say that $5 \ll 50$ is a weaker relation that $5 \ll 1000$.

**A) Cartesian product of Fuzzy sets**

Let $C = A_1 \times A_2 \times \ldots \times A_n$ be the *cartesian product* of fuzzy sets $A_1$ to $A_n$, such that if $y \in C$ and if $x_i \in U_i$ then $y = \langle x_1, x_2, \ldots, x_n \rangle$. The membership function of C is

$$Cy = (A_1 \times A_2 \times \ldots \times A_n)\langle x_1, x_2, \ldots, x_n \rangle = A_1x_1 \wedge A_2x_2 \wedge \ldots \wedge A_nx_n. \quad (2.8)$$

In other words the above expression is the minimum of all $A_i x_i$.

**B) Fuzzy Relation**

The *fuzzy relation* $R$ is the fuzzy subset of cartesian product of the universes i.e.

$$R \sqsubseteq U_1 \times U_2 \times \ldots \times U_n \tag{2.9}$$

**C) Projection**

The projection (shadow) of R on $U_{i_1} \times U_{i_2} \times \ldots \times U_{i_k}$ where $1 \le i_1 < \ldots < i_k \le n$ is the fuzzy relation

$$Proj(R; U_{i_1} \times U_{i_2} \times \ldots \times U_{i_k})\langle x_{i_1}, x_{i_2}, \ldots, x_{i_k}\rangle = \bigvee_{\substack{x_{i_t} = y_{i_t} \\ t = 1, \ldots, k}} R\langle y_1, y_2, \ldots, y_n\rangle \tag{2.10}$$

**D) Cylindrical Extension**

Let $k < n; i_1, i_2, \ldots, i_k \in \{1, 2, \ldots, n\}$ and $R \sqsubseteq U_{i_1} \times U_{i_2} \times \ldots \times U_{i_k}$ be a fuzzy relation. The *cylindrical extension* of R to $U_1 \times U_2 \times \ldots \times U_n$ is a fuzzy relation with the membership function

$$Cyl(R; U_1 \times U_2 \times \ldots \times U_n)\langle x_1, x_2, \ldots, x_n\rangle = R\langle x_{i_1}, x_{i_2}, \ldots, x_{i_k}\rangle \tag{2.11}$$

**E) Strong Composition**

Let $R \sqsubseteq U_1 \times U_2$ and $S \sqsubseteq U_2 \times U_3$ be fuzzy relations. The *strong composition* of fuzzy relations $R$ , $S$ (denoted by $R \circ S$) is the relation $R \circ S \sqsubseteq U_1 \times U_3$ with the membership function

$$(R \circ S)\langle x_1, x_3\rangle = \bigvee_{x_2 \in U_2} (R\langle x_1, x_2\rangle \wedge S\langle x_2, x_3\rangle) \tag{2.12}$$

**F) Weak Composition**

The *weak composition* of fuzzy relations $R$ , $S$ (denoted by $R \overset{\circ}{\times} S$) is the relation $R \overset{\circ}{\times} S \sqsubseteq U_1 \times U_3$ with the membership function

$$(R \overset{\circ}{\times} S)\langle x_1, x_3\rangle = \bigvee_{x_2 \in U_2} (R\langle x_1, x_2\rangle \otimes S\langle x_2, x_3\rangle) \tag{2.13}$$

where the binary operation $\otimes$ is defined by (2.6)

*Example :* Let $U_1 = \{a, b, c, d, e, f\}$ , and $U_2 = \{1, 2, \ldots, 10\}$. Define two fuzzy sets on $U_1$ and $U_2$ as

$$\begin{aligned} A &= \{0.1/a, 0.4/b, 0.9/d, 1/e\} \\ B &= \{0.9/1, 0.7/2, 0.3/5\} \end{aligned}$$

Then the Cartesian product $A \times B$ is the fuzzy set

$$\begin{aligned} A \times B = \{ & 0.1/\langle a,1\rangle, 0.1/\langle a,2\rangle, 0.1/\langle a,5\rangle, 0.4/\langle b,1\rangle, \\ & 0.4/\langle b,2\rangle, 0.3/\langle b,5\rangle, 0.9/\langle d,1\rangle, 0.7/\langle d,2\rangle, \\ & 0.3/\langle d,5\rangle, 0.9/\langle e,1\rangle, 0.7/\langle e,2\rangle, 0.3/\langle e,5\rangle\} \end{aligned}$$

Let us define a relation $R$ as

$$\begin{aligned} R = \{ & 0.1/\langle a,1\rangle, 0.4/\langle b,1\rangle, 0.6/\langle b,2\rangle, 0.2/\langle b,5\rangle, \\ & 0.9/\langle d,1\rangle, 0.7/\langle d,2\rangle, 0.8/\langle e,1\rangle, 0.7/\langle e,2\rangle, 0.1/\langle e,5\rangle\} \end{aligned}$$

Then

$$\begin{aligned} Proj(R; U_1) &= \{0.1/a, 0.6/b, 0.9/d, 0.8/e\} = R_1 \\ Proj(R; U_2) &= \{0.9/1, 0.7/2, 0.2/5\} = R_2 \\ Cyl(R_1; U_1 \times U_2) &= \bigcup_{k=1}^{10} \{0.1/\langle a,k\rangle, 0.6/\langle b,k\rangle, 0.9/\langle d,k\rangle, 0.8/\langle e,k\rangle\} \\ Cyl(R_2; U_1 \times U_2) &= \bigcup_{x \in U_1} \{0.9/\langle x,1\rangle, 0.7/\langle x,2\rangle, 0.2/\langle x,5\rangle\} \end{aligned}$$

## 2.4 Natural Language Semantics

Natural language is replete with words which describe objects, their characteristics, their intensity and relationships with other objects. We intend to utilize the principal features of natural language semantics to enable us to convert expressions involving vague terms to numbers using fuzzy set theory. This fact is highly valuable since it promises us that the computer will be able to understand man.

The lowest elements of a natural language are 'object words' whose meaning can be modelled using fuzzy sets. Typical examples of object words are "small number", "heavy-duty machine", "low speed" etc. The next category of elements are logical words like "not", "and", "or", "some", "all" etc. which serve as connectives of object words. We will interpret the logical words as *operations* with fuzzy sets. Besides these, there are terms which slightly modify the meaning of object words, like "very", "more or less", "roughly", "slightly" etc. and they will be referred to as linguistic modifiers. They can be modelled as operations on membership function.

### 2.4.1 Syntagms

A connection of words according to a preset grammatical rule, is called a *syntagm*. We will usually work with syntagms of the form **adjective-noun**. For example, a *small number*, a *heavy-duty machine*, *low speed*, *near distance*, are all syntagms. The *meaning* of syntagms will be a fuzzy set defined over an appropriate universe $U$. Consider now a set $\mathcal{T}$ of syntagms. The syntagms will be represented by the symbols $\mathcal{A}, \mathcal{B}, \ldots$. To every syntagm $\mathcal{A} \in \mathcal{T}$ and to every element $x \in U$, there is a measure expressing the degree to which the element $x$ corresponds with the syntagm $\mathcal{A}$. Thus we can define a semantic relation $S \subseteq \mathcal{T} \times U$ where $S\langle \mathcal{A}, x\rangle$ is the degree to which the syntagm $\mathcal{A}$ corresponds with the element $x \in U$. The next definition will be followed by an example to clarify this point.

**Definition :** Let S be the relation $S \subseteq \mathcal{T} \times U$ and $\mathcal{A} \in \mathcal{T}$ be a syntagm. The *meaning* $M(\mathcal{A})$ is the fuzzy set

$$M(\mathcal{A}) = \bigcup_{x \in U} S\langle \mathcal{A}, x\rangle / x \tag{2.14}$$

The meaning $M(\mathcal{A})$, $M(\mathcal{B})$, ... of syntagms $\mathcal{A}$, $\mathcal{B}$ , ... will often be denoted by the corresponding italic letters $A$,$B$,.... □

*Example :* (See [101]) Let $U =$N (set of natural numbers), and

$$\mathcal{T} = \{ \textit{small number}, \textit{big number} \}$$

. Assume the semantic relation $S$ is given by the relationship function

$$S : \mathcal{T} \times \mathcal{N} \rightarrow \langle 0, 1\rangle$$

$$
\begin{aligned}
S\langle small\ number, n\rangle &= \frac{1}{1+0.1(n-1)} \\
S\langle big\ number, n\rangle &= \begin{cases} \frac{n-1000}{n-999} & \text{for } n \geq 1000 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

where $n \in \mathsf{N}$. Then the *small number* means that it is 1 with degree 1, 5 with degree 0.71 etc. Similarly, the big number is 1005 with the degree 0.83, 2005 with the degree 1. The meaning of A will be a fuzzy set in the universe of natural numbers whose grade of membership will be calculated by the relation given by $S$. □

### 2.4.2 Linguistic modifiers

Linguistic modifiers are used in conjunction with syntagms whose meanings are either to be slightly modified or made more accurate. For example, "a very high building" , "not very clever" or "high velocity" all involve linguistic modifiers. They can be modelled in fuzzy set theory as operations on membership functions. They modify the shape of the graph of the membership function so that it is in accordance with our intuitive understanding of the notion being modelled.

The meaning of a linguistic modifier, $\mu$ is a pair of functions

$$M(\mu) = \langle \zeta_\mu, v_\mu \rangle$$

where $\zeta_\mu : U \mapsto U$ and $v_\mu : L \mapsto L$ is a function fitting the lattice $\mathcal{L}$. The function $\zeta_\mu$ depends on the type of syntagm. We will demonstrate a few of these functions in the examples that appear at the end of this section.

**Definition :** Let $\mathcal{A} \in \mathcal{T}$ be a syntagm whose meaning induces an ordering in the universe $U$. Furthermore, let $M(\mathcal{A}) = A \sqsubseteq U$ and let $\mu$ be a linguistic modifier with the meaning $M(\mu) = \langle \zeta_\mu, v_\mu \rangle$. Then the meaning of the expression $\mu\mathcal{A}$ is a fuzzy set in $U$ given by the membership function

$$M(\mu\mathcal{A})x = v_\mu(a\zeta_\mu(x)) \tag{2.15}$$

for every $x \in U$ where $a\zeta_\mu(x)$ is the membership degree of the element $\zeta_\mu(x) \in U$ in the fuzzy set $A = M(\mathcal{A})$. □

The function $v_\mu$ is usually defined by means of special functions fitting lattice $\mathcal{L}$.

**Concentration :**

$$\mathrm{CON}(\alpha) = \alpha^2 \qquad \alpha \in \langle 0,1\rangle. \tag{2.16}$$

**Dilation :**

$$\mathrm{DIL}(\alpha) = \neg\mathrm{CON}(\alpha) \qquad \alpha \in L. \tag{2.17}$$

For $L = \langle 0,1\rangle$ we obtain

$$\mathrm{DIL}(\alpha) = 2\alpha - \alpha^2 \qquad \alpha \in \langle 0,1\rangle. \tag{2.18}$$

Some linguistic modifiers will now be introduced. Let $\mathcal{A} \in \mathcal{T}$ be a syntagm with the meaning $M(\mathcal{A}) = A \sqsubseteq U$. Let

$$s_{\mathcal{A}} = \begin{cases} \inf_{x\in U}(Ker(A)) & \text{for } \mathcal{A} \text{ positive} \\ \sup_{x\in U}(Ker(A)) & \text{for } \mathcal{A} \text{ negative} \\ \text{approximate centre of } Ker(A) & \text{for } \mathcal{A} \text{ zero} \end{cases} \tag{2.19}$$

A) The modifier **very** :

It is the most common modifier whose meaning is defined as follows.

$$v_{very}(\alpha) = \mathrm{CON}(\alpha) \qquad \alpha \in L \tag{2.20}$$

$$\zeta_{very}(x) = \begin{cases} y,\ y \le x & \text{if } \mathcal{A} \text{ is positive, or } \mathcal{A} \text{ is zero and } x \le s_{\mathcal{A}} \\ y,\ y \ge x & \text{if } \mathcal{A} \text{ is negative, or } \mathcal{A} \text{ is zero and } x \ge s_{\mathcal{A}} \end{cases} \tag{2.21}$$

B) The modifier **more or less** (or **quite**):

This modifier is in some sense the inverse of the modifier *very*. Its meaning is given as follows.

$$v_{more\ or\ less}(\alpha) = \mathrm{DIL}(\alpha) \qquad \alpha \in L \tag{2.22}$$

$$\zeta_{more\ or\ less}(x) = \begin{cases} y,\ y \ge x & \text{if } \mathcal{A} \text{ is positive, or } \mathcal{A} \text{ is zero and } x \le s_{\mathcal{A}} \\ y,\ y \le x & \text{if } \mathcal{A} \text{ is negative, or } \mathcal{A} \text{ is zero and } x \ge s_{\mathcal{A}} \end{cases} \tag{2.23}$$

### 2.4.3 Linguistic variables

A syntagm describes in words the attributes or characteristics of its 'linguistic variable'. The concept of linguistic variable was introduced by L.A.Zadeh [137]. A linguistic variable is an abstract noun to which attributes are associated. For instance, when we speak of *very tall*, we refer to height. Thus *height* is the linguistic variable. In general, it is a variable whose values are words or word expressions called *terms*. The meaning of terms are fuzzy sets in the respective universes. Typical examples of linguistic variables are *age*, *truth*, *intelligence* etc. whose values (terms) may be *young*, *false*, *silly* etc. respectively.

Any variable is, in general, determined by a triple $\langle X, U, R(X)\rangle$ where $X$ is the name of the variable, $U$ is a universe and $R(X)$ is the range of the variable $X$ which is a subset of $U$, $R(X) \subseteq U$. For example, *Distance* is a variable attaining values in the universe $U = \mathsf{R}$ (real number) and $R(\mathit{Distance}) = \{1, 2, \ldots, 100\}$. The range $R(X)$ of the variable $X$ can be a fuzzy set in $U$, i.e. $R(X) \sqsubseteq U$. Then $X$ is a fuzzy variable.

**Definition** : A linguistic variable is characterised by the quintuple $\langle \mathcal{X}, \mathcal{T}(\mathcal{X}), U, G, \mathcal{M}\rangle$ where $\mathcal{X}$ is the name of the variable, $\mathcal{T}(\mathcal{X})$ is its term set, $U$ is a universe, $G$ is a syntactic rule with which the terms $\mathcal{A} \in \mathcal{T}(\mathcal{X})$ are selected, and $\mathcal{M}$ is a semantic rule assigning a meaning $M(\mathcal{A}) \sqsubseteq U$ to each term $\mathcal{A} \in \mathcal{T}(\mathcal{X})$. □

For example, consider the linguistic variable $\mathcal{X}$ = *speed*. Let us take the term set as $\mathcal{T}(\mathcal{X}) = \{$ *dead slow*, *slow*, *moderate*, *fast*, *very fast*$\}$ and the universe $U$ is on the real number line $\mathsf{R}$. We will take the syntactic rule as empty. The semantic rule $\mathcal{M}$ assigns a meaning to each term. The assessment of speed through the meaning of terms like slow, fast can not be expressed precisely through crisp numbers. It is thus more sensible to model the meaning of $\mathcal{A}$, i.e. $M(\mathcal{A})$ as a fuzzy set.

### 2.4.4 Linguistic approximation

Linguistic approximation deals with assigning meanings to elements of the term set. The meaning will be a fuzzy set having a possibility distribution over a defined universe. In a linearly ordered universe, standard functions may be employed to assign a distribution for a fuzzy set. Two such standard functions are defined below.

**Definition** : Given a linear ordering $\preceq$ of the Universe $U$, the fuzzy set $A \sqsubseteq U$ is an $S$ fuzzy set if just one of the following conditions holds for every $x, y \in U$.

$$x \preceq y \quad \text{implies} \quad Ax \leq Ay$$
$$x \succeq y \quad \text{implies} \quad Ax \geq Ay$$

If $A$ satisfies the former condition, then it is an $S^+$ fuzzy set; if $A$ satisfies the latter condition it is an $S^-$ fuzzy set.

The fuzzy set $A \sqsubseteq U$ is a $\Pi$ fuzzy set if there is a point $y_0 \in U$ such that

$$Ax \leq Ay_0 \quad \text{and} \quad Az \leq Ay_0$$

holds for every $x \preceq y_0$ and every $z \succeq y_0$. □

Let $U \sqsubseteq \mathsf{R}$ and $L\langle 0,1\rangle$. The general formula for both $S$ and $\Pi$ sets is the following where the parameters $b_1$ and $b_2$ denote inflexion points.

$$G(x, a_1, b_1, c_1, c_2, b_2, a_2) = \begin{cases} 0 & \text{if } x < a_1 \text{ or } x > a_2 \\ 1 & \text{if } c_1 \leq x \leq c_2 \\ \frac{(x-a_1)^2}{(b_1-a_1)(c_1-a_1)} & \text{if } a_1 \leq x \leq b_1 \\ 1 - \frac{(x-c_1)^2}{(c_1-b_1)(c_1-a_1)} & \text{if } b_1 \leq x \leq c_1 \\ 1 - \frac{(x-c_2)^2}{(b_2-c_2)(a_2-c_2)} & \text{if } c_2 < x \leq b_2 \\ \frac{(x-a_2)^2}{(a_2-b_2)(a_2-c_2)} & \text{if } b_2 < x \leq a_2 \end{cases} , \qquad (2.24)$$

where $x$,$a_1$,$b_1$,$c_1$,$c_2$,$b_2$,$a_2$ $\in U$. The meaning of the parameters $a_1$,$c_1$,$c_2$,$a_2$ is clear from Figure 2.1. By setting $a_1 = b_1 = c_1$ we obtain an $S^-$ fuzzy set. Similarly, $a_2 = b_2 = c_2$ yields an $S^+$ fuzzy set. For general $x$,$a_1$,$b_1$,$c_1$,$c_2$,$b_2$,$a_2$ we obtain a $\Pi$ fuzzy set.

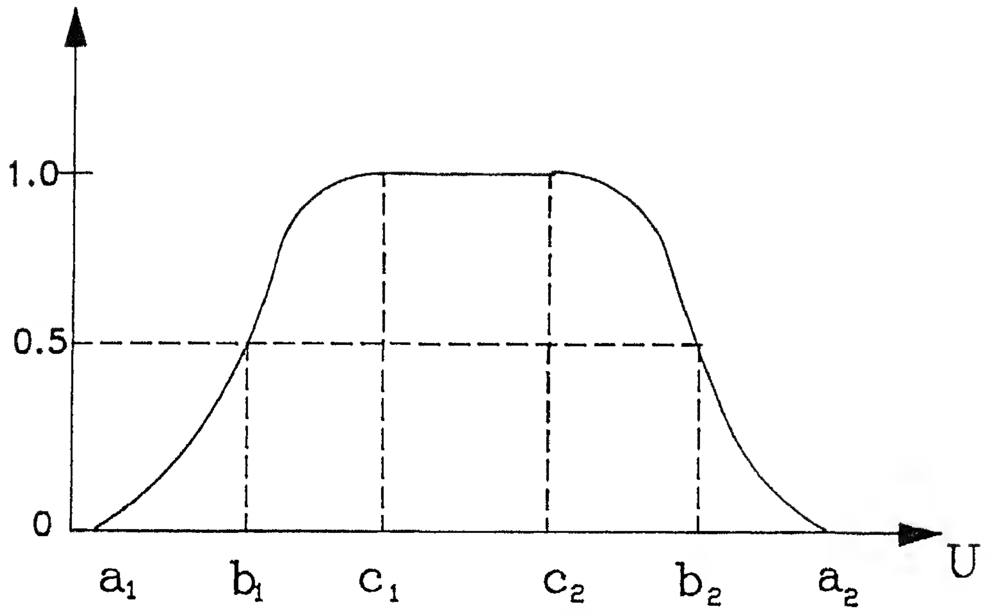


Figure 2.1: A general Π function

## 2.5 Fuzzy Control

A fuzzy controller is a linguistic description of the control process by means of conditional statements of the form

IF...THEN...

The meaning of these conditional statements can be modelled as fuzzy sets. The control algorithm is realised as approximate reasoning. It is a deterministic algorithm which is strongly non-linear, robust and if often oscillates around a stable state [101].

Let $e_j \in E_j$, $j = 1, 2, \ldots, m$ be values of the input variables and $u \in U$ be values of an output variable. Assume we are given the linguistic variables

$$\begin{aligned} \mathcal{X}_{E_j} &= \text{the } j^{th} \text{ variable} \\ \mathcal{X}_U &= \text{the output variable} \end{aligned}$$

with the corresponding universes $E_j$, $U$, $j = 1, 2, \ldots, m$.

The fuzzy controller is the statement

$$\mathcal{R} = \mathcal{R}_1 \text{ ELSE } \ldots \text{ ELSE } \mathcal{R}_n \tag{2.25}$$

where each $\mathcal{R}_i$, $i = 1, 2, \ldots, n$ is the conditional statement (also call *rule*),

$$\mathcal{R}_i = \text{IF } e_1 \text{ is } \mathcal{A}_{E_1,i} \text{ AND } \ldots \text{ AND } e_m \text{ is } \mathcal{A}_{E_m,i} \text{ THEN } u \text{ is } \mathcal{A}_{U,i} \tag{2.26}$$

and the terms $\mathcal{A}_{E_j,i}$, $j = 1, 2, \ldots, m$ and $\mathcal{A}_{U,i}$ belong to the term sets $\mathcal{T}(\mathcal{X}_{E_j})$ and $\mathcal{T}(\mathcal{X}_U)$ respectively. The conditional statements $\mathcal{R}_i$ (2.26) are understood to be linguistic expressions of logical implication. We take the control algorithm to be

$$M(\mathcal{R}_i) = (A_{E_1,i} \times A_{E_2,i} \times \ldots \times A_{E_m,i})^* \circledast \; A^*_{U,i} \tag{2.27}$$

where by the symbol "$*$" we mean the cylindrical extension of these fuzzy relations to $E_1 \times E_2 \times \ldots \times E_m \times U$ (see 2.11). The meaning of $\mathcal{R}$ is given by the intersection

$$M(\mathcal{R}) = \bigcap_{i=1}^{n} M(\mathcal{R}_i) \tag{2.28}$$

If we are given the actual input values $\widehat{A}_{E_j}$, $j = 1, 2, \ldots, m$ then the magnitude of the output is computed from the actual input values using the weak composition (2.13) of fuzzy relations as follows.

$$\widehat{A}_U = (\widehat{A}_{E_1} \times \widehat{A}_{E_2} \times \ldots \times \widehat{A}_{E_m}) \overset{\circ}{\times} M(\mathcal{R}) \tag{2.29}$$

From the resulting fuzzy set $\widehat{A}_U$ we must determine the crisp value of $\widehat{u} \in U$. The action $\widehat{u}$ is determined by means of the centre of gravity under the menbership grade curve $\widehat{A}_U$.

$$\widehat{u} = \frac{k_R \int_U u \, d\widehat{A}_U}{\int_U d\widehat{A}_U} \tag{2.30}$$

where $k_R$ is a proportionality constant depending on the problem. For some cases, heuristic methods depending on the particular problem might be perhaps better for determining crisp values of $\widehat{u}$. The process of generating the crisp value of a variable from its fuzzy representation, is called *defuzzification*.

## 2.6 Remarks

The aim of presenting fuzzy set theory in this chapter was to build up the background for developing a fuzzy controller. Most of the rigourous mathematics will not be referred later, but in the following chapter we pick up a case and elaborate how we can develop a fuzzy controller for a motion planning problem with collision avoidance. In the succeeding chapters we demonstrate how it has been applied to three different problems.

*The good rule*
*Sufficeth them, the simple plan,*
*That they should take,*
*who have the power*
*And they should keep who can.*
– WILLIAM WORDSWORTH

# Chapter 3

# The Application of Fuzzy Logic to Motion Planning

## 3.1 Introduction

This chapter presents a discussion on how the motion planning problem can be cast as a fuzzy control problem. It demonstrates that collision avoidance strategies can be posed as simple fuzzy *rules*. A natural outcome of these rules is a response which is similar to one commonly followed in life. Particular instances of motion planning are explained in detail in later chapters.

Motion planning pertains to those aspects which involve moving an *object* or a group of *objects* from one location to another. The *objects* may be individual bodies that move independently within the work volume like mobile platforms, or they may be connected bodies which must follow a definite kinematic relationship between themselves, like the links of a robot arm. Important aspects in motion planning are finding the optimal path between the goal points of the bodies, or finding collision free configurations of the objects in the environment, or both. The emphasis of this thesis is only on finding collision free configurations for objects.

Structured environments are those in which the motions of all bodies are defined precisely and well known *a priori*. An algorithm that generates collision free configurations for bodies in a structured environment, will yield exact solutions. However, in an unstructured environment, which is more common, the motion of bodies is unknown. Thus an algorithm which works in an unstructured environment must have some sensing capability to *feel* or *see* the environment. Since distance measurement is the primary objective, it is decided to use some kind of ranging sensor. A brief discussion on sensors and their accuracy is also presented in this chapter.

The accuracy of ranging sensors poses a big problem for its practical utilization. Accurate ranging sensors are expensive which prohibits their use in sensing large volumetric spaces in a complicated environment since large number of sensors would be required. Thus inexpensive sensors must be used in large quantities to sense complicated environments sacrificing high accuracy. In fact, for many applications like avoiding collision between approaching vehicles where the sizes of the objects involved are large, too much accuracy is not needed. A robust algorithm must be able to *smoothen* out the inaccuracies in measured data so that it can be reliably used within a specified safety margin. To evolve a robust algorithm and to cope with inaccuracies, fuzzy logic has been used. The motion commands are generated through a fuzzy controller. This chapter also explains how instantaneous motion can be planned using fuzzy rules.

Finally, a simple example is used to illustrate the basic features of the fuzzy algorithm. Without mentioning specific applications, we describe how the set of fuzzy variables is chosen, how the elements of term sets are modelled and how the output parameter is converted to a motion parameter.

## 3.2 Use of Sensors in Motion Planning

Sensors are necessary to give warning signals about approaching bodies in an unstructured environment. Sensors are mounted at appropriate locations on the body from where sensing poses an advantage. For example, in a vehicle they may be mounted around the periphery to detect approaching bodies. In planar motion,

CENTRAL LIBRARY
I.I.T. KANPUR
Acc. No. A. 116567

a sensor which directs its beam perpendicular to the plane of motion is meaningless because collision is unlikely to occur from these directions. If sensors are mounted on a robot arm to protect it from collision against incoming bodies, they have to cover spaces which are most likely to be occupied by foreign bodies.

Distance measurement can be achieved through either ranging sensors or proximity sensors. Generally speaking, ranging sensors measure large distances while proximity sensors measure smaller distances. Thus, range sensing provides gross guidance information to the object while proximity sensing helps in fine motion planning, such as during the terminal stages of object grasping. Since gross motion planning is one of the primary concerns, only range finding sensors are considered here. Broadly, there are three kinds of techniques for range sensing. They are

- Triangulation approach
- Structured lighting approach
- Time-of-flight range finding

Among the three, our primary focus is on time-of-flight ranging sensors because its requires least processing making it considerably faster. This technique involves sending a pulse over a short time and then receiving an echo from a nearby object. The time duration between the outgoing pulse and the return echo will give an estimate of the distance to the reflecting surface. The signal could be a pulsed laser, a continuous laser or an ultrasonic beam. Ultrasonic sensors are quite inexpensive and can be used in large quantities, though having inherent problems of inaccuracy.

For a typical ultrasonic sensor (manufactured by Polaroid), a 1 millisecond chirp, consisting of 56 pulses at four frequencies, 50, 53, 57, 60 KHz, is transmitted by a transducer $1\frac{1}{2}$ inches in diameter. The signal reflected by object is detected by the same transducer and processed by an amplifier and other circuitry capable of measuring distance from 0.9 feet to 35 feet with an accuracy of about 1 inch. The mixed frequencies in the chirp are used to reduce signal cancellation. The beam pattern of this device is shown in Figure 3.1. It is found that the sensor will be most effective in the range $\pm 30°$. This imposes severe limitations in its resolution if it is to be used for accurate measurements.

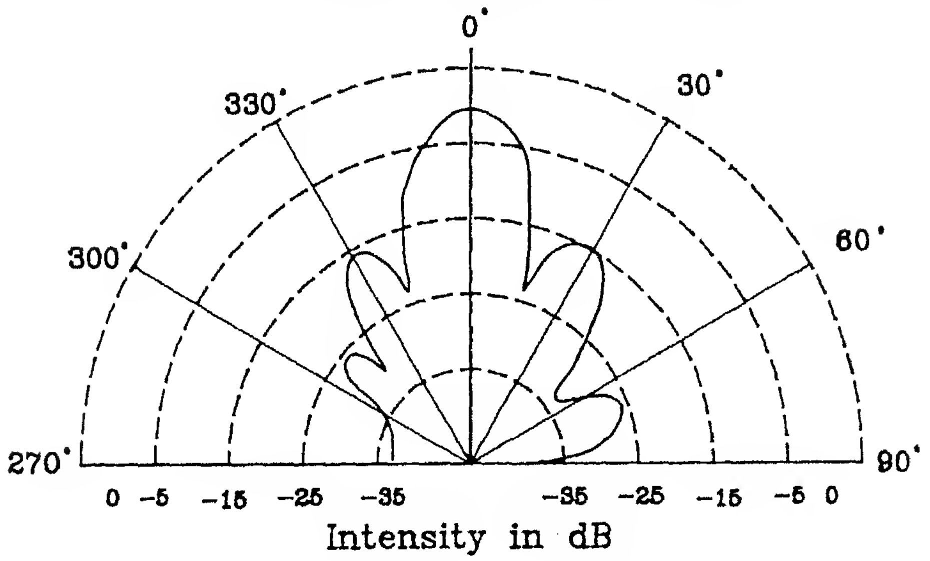


Figure 3.1: Typical beam pattern of an ultrasonic sensor at 50 kHz

Accurate sensors are meaningful where precision is required. High precision entails large computation which in turn makes the system slower and renders it unusable for on-line computation. In fact for most practical motion planning problems, too much accuracy is not needed. So ultrasonic sensors are best used in navigation problems and for collision-free manipulator guidance.

Our proposal is thus to use inaccurate and cheap sensors so that a large number of sensors can be used without compunction. The inaccuracy of the sensors will be taken care of by a robust algorithm. Since too much processing will not be required, it will make the system faster and appropriate for on-line use.

## 3.3 Approximate Approach to Motion Planning

To most of us, motion planning is an inherent aspect of our constitution that we unconsciously practice throughout our life. Routine instances of avoiding collision with walls, moving vehicles or other static or dynamic objects are all examples of motion planning. To make the point clear, a very common example is taken.

Let us assume that a driver wishes to take his vehicle from a starting location to a goal location. We focus attention on his strategy of collision avoidance with oncoming vehicles, while he is steering on the road. When the driver sees an oncoming vehicle, he first *estimates* the distance, speed, and direction of approach of the vehicle. This estimate is based on his driving experience. From his estimates of these parameters, he makes a *guess* of an impending collision. Based on his guess he adopts a collision *avoidance* strategy. What are the numerous options he can follow ? He may, for example, change his speed without changing direction. He may also change his direction by steering away at the same speed. Alternately, he can both change his speed and direction. Perhaps, he is following a rule which instructs, "*Steer your own vehicle to the left when you see another vehicle near you.*"

Viewed from the perspective of fuzzy set theory, the driver's fuzzy variables are distance, angle and speed. The driver's estimates of distances and angles are in *terms* of "near", "far", "left", "right", "fast" etc. but definitely not as 13.24 meters or $24\frac{3}{4}$ degrees. Thus the driver has assigned vague *meanings* to the terms describing estimates of distance and angle. It is thus possible to model the meaning of the terms as a fuzzy set. His strategy of collision avoidance is also based on rough statements like "turn slightly to the left". This statement (or rule) is a result of his driving experience. We thus conclude that the driver's experience is the fuzzy controller that guides him on the road. In this work, we intend to obtain the estimates of distance and speed with the help of ultrasonic sensors which are not very reliable and accurate. To smoothen out inaccuracies in sensory data the input is fuzzified, and then fed it into the fuzzy controller. Avoidance measures will be given in terms of fuzzy rules. The fuzzy rules are formulated in such a way that instructs one or both bodies to move away from each other when distances are sensed to be "very near". The output of the fuzzy controller will be defuzzified to convert to an actual motion command. A formal setup to formulate the fuzzy rules is presented in the following section with an example.

## 3.4 Motion Planning in Terms of Fuzzy Control

Motion planning will, broadly speaking, involve estimating the relationship between two bodies in terms of distance, angle or speed of approach. Thus, a methodology dealing with these parameters will be general enough to encompass any kind of collision avoidance problem.

### 3.4.1 Choice of set of fuzzy variables

The set of fuzzy variables must naturally have its elements as *distance*, *angle* and *speed.* Although *speed* is being referred as a separate variable, it is actually a time derivative of distance. An estimate of speed is thus possible by monitoring distance over a period of time. Hence, it is decided to exclude it from the variable set. It will be shown later in Section 3.5 that speed can be calculated by using *Distance* as the base variable. Hence, in the language of Fuzzy Set Theory, the variable set $\mathcal{X}$ is,

$$\mathcal{X} = \{Distance,\ Angle\} \tag{3.1}$$

In the variable set

$$\begin{aligned} \mathcal{X}_1 &= Distance \\ \mathcal{X}_2 &= Angle \end{aligned}$$

### 3.4.2 Definition of appropriate fuzzy quantities

#### Defining universe

A Universe is now defined for each of the elements in $\mathcal{X}$ and are designated as $\mathbf{U}_D$ and $\mathbf{U}_A$, respectively belonging to the real number line. The most natural choice for a Universe would be to assign discrete values for each ranging from the smallest possible measure to the largest one, with $\mathrm{U}_k \in \mathrm{R}$. After some experimentation it was found that, in case of *Distance*, it is advantageous to have a finer gradation between the elements when distances are small. The reason behind this choice is that control should be more accurate when distances are small i.e. when collision is most likely. Thus the following definition of Universe of Distance was found satisfactory.

$$\mathbf{U}_D = \left\{x_i; x_i = \frac{i(i-1)\mathrm{H}_D}{\mathrm{N}_D(\mathrm{N}_D-1)}, 1 \leq i \leq \mathrm{N}_D\right\} \tag{3.2}$$

where $H_D$ is the largest measurable distance and $N_D$ is the cardinal number of $U_D$. Thus, the elements of $\mathbf{U}_D$ for $H_D = 105$ and $N_D = 15$ are

$$U_D = \{0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105\} \tag{3.3}$$

The Universe of *Angle* is distributed uniformly over the range $\Phi_{min}$ to $\Phi_{max}$, where $\Phi_{min}$ is the lower limit of the angle range and $\Phi_{max}$ is its upper limit. Thus

$$\mathbf{U}_A = \left\{x_i; x_i = \Phi_{min} + \frac{\Phi_{max} - \Phi_{min}}{N_A}(i-1), 1 \leq i \leq N_A\right\} \tag{3.4}$$

where $N_A$ is the cardinal number of $\mathbf{U}_A$. For example, if $\Phi_{min} = 0°$, $\Phi_{max} = 360°$ and $N_A = 8$, the elements of the universe of angle are

$$U_A = \{0, 45, 90, 135, 180, 225, 270, 315\} \tag{3.5}$$

The choice of $N_A$ is governed by the beam angle of the sensors.

### Term sets and its meaning

The term sets, $\mathcal{T}(\mathcal{X}_i)$, of distance $\mathcal{X}_1$ and angle $\mathcal{X}_2$ for the Universe $\mathbf{U}_A$ defined in (3.5) are enumerated as

$$\begin{aligned} \mathcal{T}(\mathcal{X}_1) &= \{VeryNear, Near, QuiteNear, ModerateDistance, \\ &\quad Far, VeryFar\} & (3.6) \\ \mathcal{T}(\mathcal{X}_2) &= \{East, North\text{–}East, North, North\text{–}West, \\ &\quad West, South\text{–}West, South, South\text{–}East\} & (3.7) \end{aligned}$$

A choice of eight directions is made because it is easier to comprehend and they are commonly used in navigation. This is helpful later when defining the fuzzy rules for collision avoidance[1].

[1]A choice of eight directions of movement has also been proposed by other researchers [3,139] for movements of bodies.

### 3.4.3 Fuzzy representation of linguistic descriptors

The syntactic rules for generating the meaning of the terms (e.g. "*Near*", "*North*") can be conveniently expressed by either the $\Pi$ or $S$ fuzzy sets (2.24) as described in Section 2.4.4. For instance, the meaning of the quantity "*Near*" for the Universe $\mathrm{U}_D$ defined in (3.3) may be assigned by

$$M(Near) = \bigcup_{x \in \mathrm{U}_D} G(x, 0, 0, 0, 0, 12.5, 25) \tag{3.8}$$

in which "*Near*" represents a $S^-$ fuzzy function. Similarly

$$M(Far) = \bigcup_{x \in \mathrm{U}_D} G(x, 50, 75, 105, 105, 105, 105) \tag{3.9}$$

is a $S^+$ fuzzy function. The linguistic modifiers can be used to define additional terms, for example

$$\begin{aligned} M(Very\,Near) &= CON(M(Near)) && (3.10)\\ M(Quite\,Near) &= DIL(M(Near)) && (3.11) \end{aligned}$$

A pictorial representation of the above quantities is given in Figure 3.2. The abscissa of the graph represents distance, while the ordinate represents the membership value varying from 0.0 to 1.0. It is seen that the terms "*Near*" "*Very Near*" and "*Quite Near*" originate with a membership value of 1.0 at distance 0, and fade away to 0.0 at higher values of distance. On the other hand, the term "*Far*" has a membership value of 1.0 at $\mathrm{H}_d$ and fades to 0.0 for lower values of distance. The term "*Moderate Distance*" has the highest membership near $\mathrm{H}_d/2$.

All the elements of $\mathcal{T}(\mathcal{X}_2)$ (i.e. term set of angle) are defined as standard $\Pi$ functions, e.g. *North* is modelled as

$$M(North) = \bigcup_{x \in \mathrm{U}_A} G(x, 30, 60, 90, 90, 120, 150) \tag{3.12}$$

where the numeric values represent angles in degrees. The membership grade curve of the above terms are pictorially represented as shown in Figure 3.3. As earlier, the ordinate represents the membership value while the abscissa represents the value of angle in degrees.

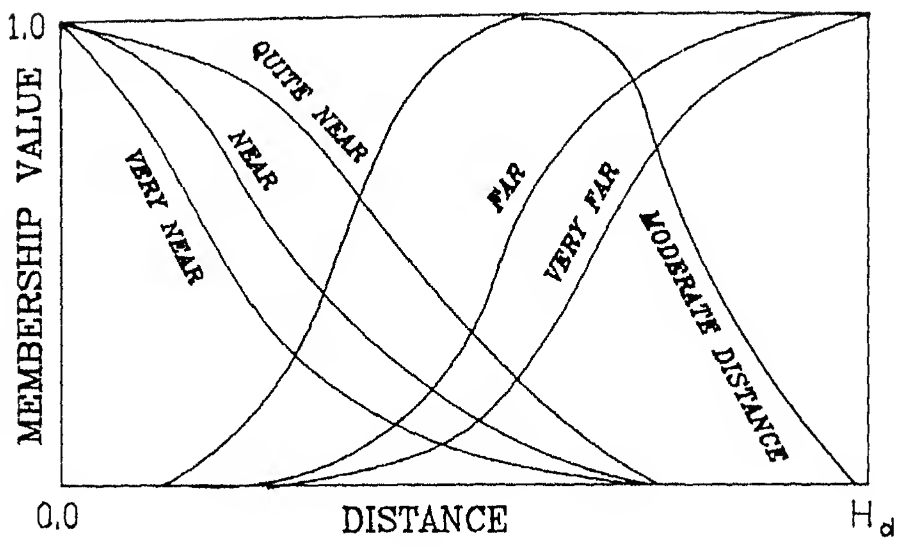


Figure 3.2: Meanings of terms representing distance

## 3.5 Demonstration of Fuzzy Control

In this section the application of the fuzzy compositional rule of inference is demonstrated through an example. A fuzzy controller is described in the form of rules. The number of control variables is restricted to two in this thesis, since it is cumbersome to tackle more than two variables at a time due to large memory requirements. A simple set of rules is taken to illustrate how the input quantities are fuzzified and fed into the controller. The output, being a fuzzy quantity, is

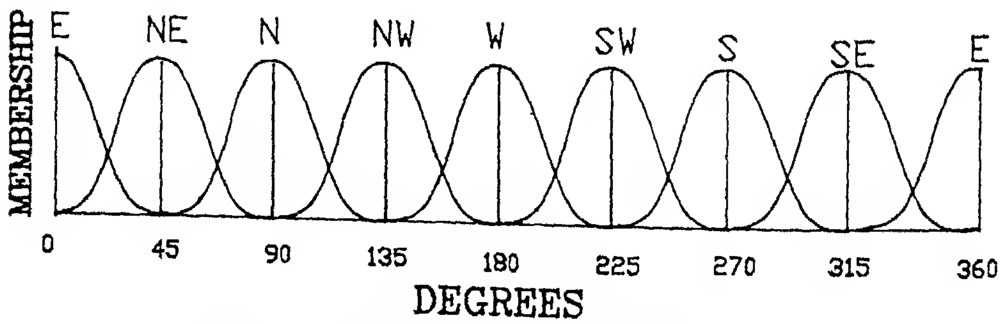


Figure 3.3: Meanings of terms representing angle

defuzzified and the resulting crisp value is interpreted as a motion command.

The interpretation of the fuzzy input variables depends on the problem formulation. Let us first propose a simple set of rules to demonstrate our point.

```
Rule 1 : if ((Estimated Distance is Near) and
           (Estimated Angle is North)) then
                                      (Move East)
Rule 2 : if ((Estimated Distance is Very Near) and
           (Estimated Angle is North-West)) then
                                      (Move North-East)
```

These rules employ quantities like "`Estimated Distance`" and "`Estimated Angle`" which are given the values "`Near`" and "`North`" in the first rule and "`Very Near`" and "`North-West`" in the next rule. The interpretation of `Distance` and `Angle` is subject to change depending on the problem. For the above set of rules, `Estimated Distance` obviously refers to the separation between two bodies while `Estimated Angle` refers to the angular position of one body with respect to another. Without breaking continuity, we may presently ignore the contextual relevance of terms `Distance` and `Angle` and assume them to be just input quantities for a fuzzy controller. Later in Chapter 4, it is shown that a different interpretation of Angle is also possible.

In this section the set of rules given above are considered leaving the interpretation of rules to later chapters where specific problems are discussed. These rules serve only as examples, and are subject to change depending on the problem. The actual rules used to solve some of the problems are given in Chapters 4, 5 and 6 where specific problems are discussed.

First, the meaning of terms appearing in the fuzzy rules are defined. These are the terms `Near` and `Very Near` defined over the universe `Distance`, and `North`, `East`, `North-East` and `North-West` over the Universe `Angle`. The meaning of these terms are assigned according to the procedure outlined in Section 2.4.4 using the formula given in (2.24). For the Universes defined in (3.3) and (3.5) and the meaning of term sets defined as in (3.8) and (3.12), the following fuzzy sets are obtained.

$$
\begin{aligned}
Near &= \left\{\frac{1}{0},\frac{0.9968}{1},\frac{0.9711}{3},\frac{0.8848}{6},\frac{0.68}{10},\frac{0.32}{15},\frac{0.0512}{21}\right\}\\
Very\ Near &= \left\{\frac{1}{0},\frac{0.9936}{1},\frac{0.9430}{3},\frac{0.7829}{6},\frac{0.4624}{10},\frac{0.1024}{15},\frac{0.0026}{21}\right\}\\
North &= \left\{\frac{0.125}{45},\frac{1}{90},\frac{0.125}{135}\right\}\\
East &= \left\{\frac{1}{0},\frac{0.125}{45},\frac{0.125}{315}\right\}\\
North\text{–}East &= \left\{\frac{0.125}{0},\frac{1}{45},\frac{0.125}{90}\right\}\\
North\text{–}West &= \left\{\frac{0.125}{90},\frac{1}{135},\frac{0.125}{180}\right\}
\end{aligned}
$$

The fuzzy rules given earlier, when translated to a computer representation, yields an array of dimension $N_D \times N_A \times N_A$. This three dimensional array, is reproduced below for each slice. The elements are obtained using the compositional rule given in (2.27). Eight slices from the resulting matrix of Rule 1 is given below. The reader is encouraged to check the values for a few elements.

$$
Matrix_{i,j,1} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

$$Matrix_{i,j,2} = \begin{bmatrix} 1 & 1 & 0.125 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.1282 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.1539 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.2402 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.445 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.805 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,3} = \begin{bmatrix} 1 & 0.875 & 0 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0032 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0289 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.1152 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.32 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.68 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.9488 & 0.9488 & 0.9488 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,4} = \begin{bmatrix} 1 & 0.875 & 0 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0032 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0289 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.1152 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.32 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.68 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.9488 & 0.9488 & 0.9488 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,5} = \begin{bmatrix} 1 & 0.875 & 0 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0032 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0289 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.1152 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.32 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.68 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.9488 & 0.9488 & 0.9488 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,6} = \begin{bmatrix} 1 & 0.875 & 0 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0032 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0289 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.1152 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.32 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.68 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.9488 & 0.9488 & 0.9488 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,7} = \begin{bmatrix} 1 & 0.875 & 0 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0032 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.0289 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.1152 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.32 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.875 & 0.68 & 0.875 & 1 & 1 & 1 & 1 \\ 1 & 0.9488 & 0.9488 & 0.9488 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Matrix_{i,j,8} = \begin{bmatrix} 1 & 1 & 0.125 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.1282 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.1539 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.2402 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.445 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.805 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The elements from left to right are over the Universe `Angle` and from top to bottom are over the Universe `Distance`. Each element of the matrix for rule 1 turns out to be

$$Matrix_{i,j,k} = \min(1, 1 - min(Near\ x_i, North\ x_j) + East\ x_k)$$

For example the element

$$\begin{aligned} Matrix_{2,3,4} &= \min(1, 1 - \min(0.9968, 1) + 0) \\ &= 0.0032, \end{aligned}$$

which can be easily checked from the given matrix.

The application of the fuzzy rules to yield an output parameter from actual input parameters is now demonstrated. Let us suppose that the sensory data indicates that measured `Distance` is 10 units and measured `Angle` is 110° for a particular instance. We have to infer the output parameter from the fuzzy rules.

The first step is to fuzzify the input parameters. A dispersion (or *bandwidth*[2]) of 60 units for distance, and 120° for angle is assumed. Thus the fuzzified distance is

$$Distance_{observed} = \left\{\frac{0.777}{0}, \frac{0.8199}{1}, \frac{0.8911}{3}, \frac{0.9644}{6}, \frac{1}{10}, \frac{0.9444}{15}, \frac{0.7311}{21}, \frac{0.3200}{28}, \frac{0.0355}{36}\right\}$$

while the fuzzified angle is

$$Angle_{observed} = \left\{\frac{0.7777}{90}, \frac{0.6527}{135}\right\}.$$

The application of Rule 1 using the weak composition given in (2.13) gives the fuzzified output as

$$Angle_{output,1} = \left\{\frac{0.7777}{0}, \frac{0.7311}{45}, \frac{0.6799}{90}, \frac{0.6799}{135}, \frac{0.6799}{180}, \frac{0.6799}{225}, \frac{0.6799}{270}, \frac{0.7311}{315}\right\}. \quad (3.13)$$

Similarly, the application of Rule 2 gives the fuzzified output as

$$Angle_{output,2} = \left\{\frac{0.7777}{0}, \frac{0.7777}{45}, \frac{0.7777}{90}, \frac{0.7285}{135}, \frac{0.7285}{180}, \quad (3.14)$$
$$\frac{0.7285}{225}, \frac{0.7285}{270}, \frac{0.7285}{315}\right\}. \quad (3.15)$$

According to (2.28), the intersection of the fuzzy sets in (3.13) and (3.14) yields the output fuzzy parameter

$$Angle_{output} = \left\{\frac{0.7777}{0}, \frac{0.7311}{45}, \frac{0.6799}{90}, \frac{0.6799}{135}, \frac{0.6799}{180}, \frac{0.6799}{225}, \frac{0.6799}{270}, \frac{0.7285}{315}\right\}.$$

At this stage the fuzzy output parameter is de-fuzzified to get the resulting crisp output parameter. The defuzzification is done on the basis of (2.30) with $k_R$ taken as 100. The resulting output vector $\vec{\Phi}$ is obtained as

$$\vec{\Phi} = 16.8359\,\vec{\imath} + 0.18382\,\vec{\jmath}$$

[2]The quantity $(a_2 - a_1)$ in (2.24) is called the *bandwidth* of the fuzzy set

where $\vec{i}$ and $\vec{j}$ are the unit vectors along the $x$ and $y$ axes respectively. The vector $\vec{\Phi}$ may be interpreted, for example, as the distance by which an object must shift in order to avoid collision. Thus the object must shift by 16.8369 units with an angle of about 1° from the X-axis, which is almost *East.* This is in accordance with Rule 1 since the input value of the Angle was 110°, which is almost *North.* In our example, Rule 1 has taken predominance over Rule 2.

It is interesting to note that a single fuzzy output parameter representing angle, has yielded a *vector* quantity involving two parameters — direction and magnitude. This is because the inputs have been taken to be normal fuzzy sets (see page 36). The output parameter on the other hand has no such restriction. Thus the maximum grade of membership in the output parameter, which is always less than 1, is a measure of the *strength* of the output. In other words, an extra parameter is *hidden* inside the output which, if extracted, can lead to a vector quantity, involving *both* magnitude and angle. The magnitude is made proportional to height (see page 36) of the output variable whereas the angle can be deduced from the location of the centre of gravity along the universe. This is the reason why *Speed* has not been included in the set of fuzzy variables. The extra parameter in the fuzzy output can, for example, be used to interpret *speed* by which the two bodies have to separate in order to avoid collision. The above strategy results in considerable saving of computation time as well as reduces memory requirements.

It is clear that the robustness of the algorithm can be increased by increasing the *number* of rules, but at the same time sacrificing speed of computation. In the following chapters examples are taken to demonstrate use of this algorithm for motion planning of multiple bodies moving independently on a plane, then for motion planning of a two-dimensional redundant manipulator moving amongst unknown moving obstacles and finally for motion planning of a three-dimensional manipulator arm moving among unknown moving obstacles.

*For my part, I travel*
*not to go anywhere, but to go.*
*I travel for travel's sake.*
*The great affair is to move.*
[Travels with a Donkey]
– ROBERT LOUIS STEVENSON

# Chapter 4

# Collision Avoidance Strategies for Planar Navigation Problems

## 4.1 Introduction

In this chapter the problem of collision avoidance of bodies moving on a plane is considered. Instances of this situation are common in navigation. First the problem of collision avoidance of stationary bodies with only one moving body is considered, and it is later extended for multiple moving bodies. In both problems, prior knowledge about the bodies is assumed to be unknown. Avoidance measures for bodies are found based on locally perceived instantaneous information. It is assumed that sensors provide feedback about two important parameters — the proximity of an approaching body and its direction of approach.

## 4.2 Problem Statement

A set of bodies $B_i$, $i = 1, 2 \ldots, n$ are considered which are free to move on a plane. The initial positions of the bodies are denoted by $B_i^S$ and the goal positions by $B_i^G$ (see Figure 4.1). The current position of body $B_i$, its start position $B_i^S$, and its

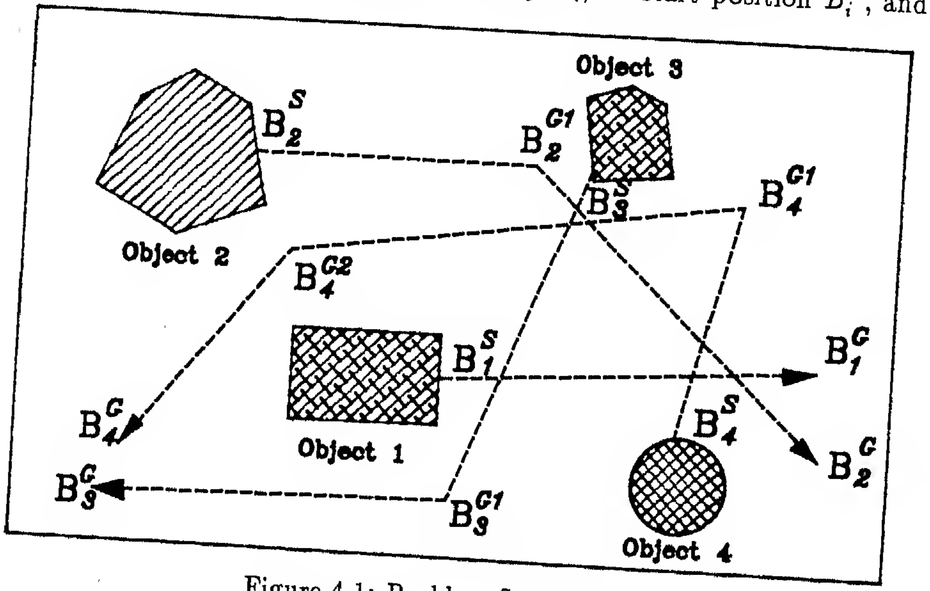


Figure 4.1: Problem Statement

goal position $B_i^G$ is known to the controller of $B_i$ during execution. The presence, motion, start or goal position of any other body $B_j, j \neq i$ is completely unknown to the controller of $B_i$ before the start of execution. We wish to investigate methods of guiding each body to its goal without colliding with neighboring bodies.

The interaction between two bodies will be assessed on the basis of sensor readings taken on-line. At this stage we are not concerned with the mode of sensor installation on an object, but it is assumed that two parameters necessary for decision making, the distance of an obstacle from the periphery of a body and the angle of approach of the obstacle are known to us. A scheme to install sensors around an object is suggested in Section 4.7. Since distances are measured from the *periphery* of the bodies, the *shape* of the body is not important in our discussion. A simplification is thus introduced which leads to axi-symmetry by assuming all bodies to be of circular shape. The size of the circle representing an object is chosen as the smallest circle enclosing the entire body. The position of the body is found from the location of the centre of the circle, and the path traced by the body is the *locus* of the centre of the circle. Figure 4.2 shows an

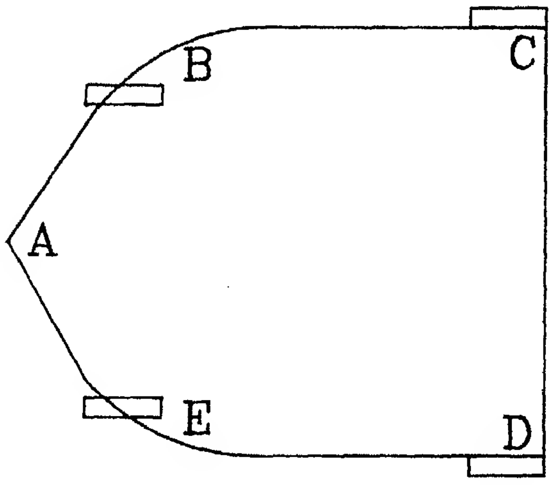


Figure 4.2: An arbitrary shaped body (representing a vehicle)

arbitrary shaped body. Figure 4.3 shows the enclosing circle of the body shown in Figure 4.2. The avoidance measures are found on the basis of only two translational parameters — the $x$ and $y$ components of position vector, but while deciding the actual motion of the bodies the rotational parameter is also considered. Although this is an incomplete representation, it is shown that it is still possible to solve quite complex problems. A safe path calculated for the enclosing circle will obviously also be safe for the actual body it encloses.

For the sake of categorization the moving bodies are sub-divided into two classes — bodies that have to follow a prescribed path, and bodies that do not have a prescribed path. Bodies of the first kind will be called "**path-preference** bodies", and bodies of the second kind "**point-preference** bodies". Bodies that fall in the second category have to *touch* a set of $p$ points in between $B_i^S$, $B_i^{G1}$, $B_i^{G2}, \ldots, B_i^{G(p-1)}$, $B_i^{Gp}$, $B_i^G$ in the prescribed sequence while they are on their way to the final goal point $B_i^G$.

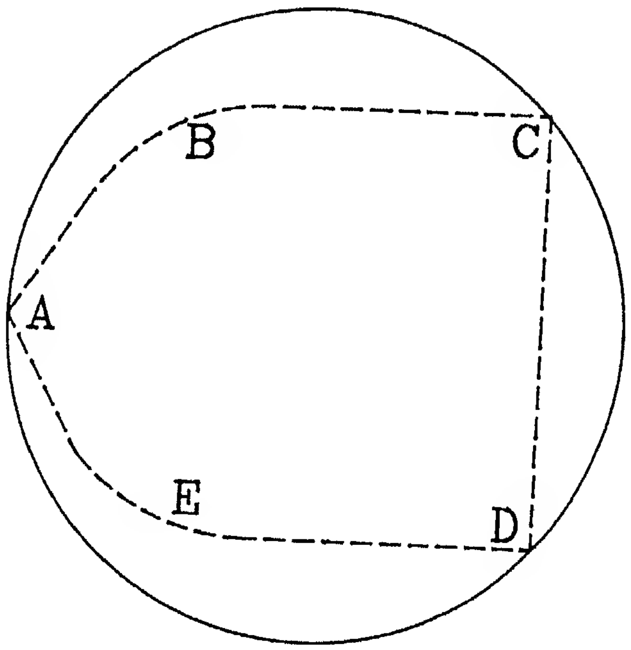


Figure 4.3: Representing body as a circle

## 4.3 Fuzzy Rules for Two-Body Interaction

### 4.3.1 Outline of fuzzy algorithm

Our aim in this chapter is to come up with strategies for maneuvering multiple moving bodies on a plane. Before this problem is taken up, a simpler problem is considered to demonstrate the applicability of the fuzzy rules. This problem is termed the "Path Clearance Problem" and involves the interaction to two bodies — one which is moving, and the other which is initially static but is allowed to shift to a safe location under the influence of the moving body.

The body for which motion planning is done (i.e. the body which is static initially and which is the focus of our attention) is called the "**primary body**". The other body which is moving is perceived as an obstacle by the *primary body*, and is called the "**secondary body**". The primary body will be symbolically represented as **P**, while the secondary body as **S**. Rules will be developed for the primary body to avoid collision with the secondary body if it comes dangerously near. Our attempt will be to move the primary body so that it avoids collision with the secondary body. Presently we are not interested in the path-planning of the moving body.

The coordinate system necessary for establishing the position of a body inside a frame is now described. Figure 4.4 shows a plane whose reference coordinate system is shown as $X_0$-$Y_0$. As referred in Section 3.4.1, eight fuzzy directions are chosen to represent angle. The terms representing the fuzzy directions are given in (3.6). The periphery of each circular body is thus divided into eight segments, and they are numbered sequentially from 1,...,8 as shown in Figure 4.4. The

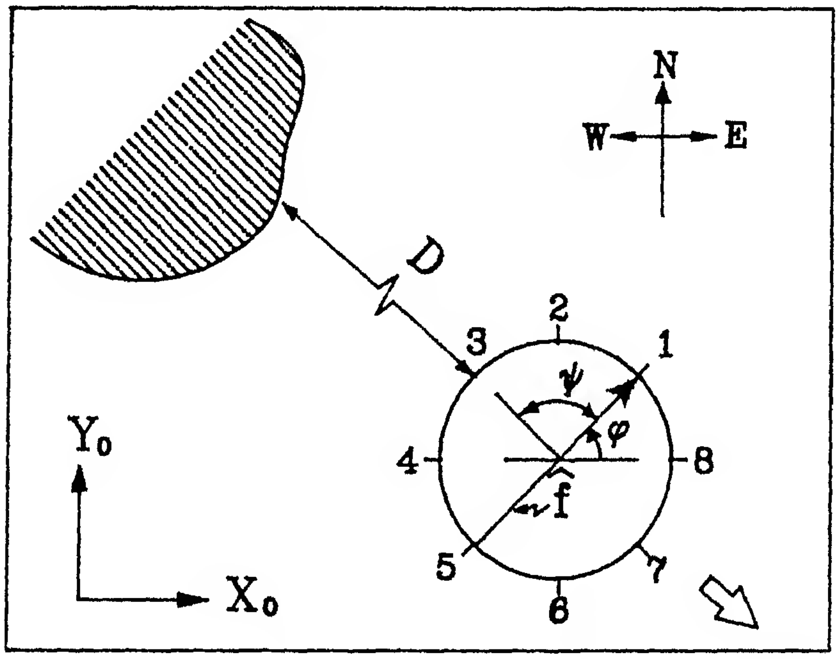


Figure 4.4: Segmenting the outer boundary of an object into eight sections

arrow from location 5 to 1 symbolically shows the direction of the **front** of the body. This vector is represented as $\hat{f}$ in Figure 4.4. The arrow head will always point towards sensor 1. In Figure 4.4 the body is pointing towards North-East of the frame $X_0$-$Y_0$. Let us now suppose that the sensors have detected an obstacle near the body. This obstacle is shown by the shaded region in Figure 4.4. The assessment of exact position of the obstacle relative to the position of the primary body will depend on the arrangement of sensors. We do not intend to investigate this issue right now, but later in Section 4.7 a simple proposal to solve this problem is presented. Let us assume for the present that we have been able to assess that the obstacle in Figure 4.4 has been detected to be *somewhat* "North-West" of the primary body.

Let us focus attention on the fuzzy rules that are to be applied for avoiding collision with the obstacle. The application of the rules will suggest a direction of movement to the primary body for evading the secondary body. The resulting motion of the object is called **escape motion** and the direction of motion as **escape direction**. The escape motion will be represented by an **escape vector**, $\vec{v}_{B_i, Esc}$. The rules will be of the form described in (2.25) and (2.26). Thus a typical set of rules may be written as

```
if (Estimated Distance of S is NEAR) and
     (Estimated Relative Angle of S with respect toP is SOUTH)
      then (move P NORTH)
if (Estimated Distance of S is NEAR) and
     (Estimated Relative Angle of S with respect toP is SOUTH-EAST)
      then (move P NORTH-WEST)
⋮
... and so on.
```

These rules are laid down here only for illustration purpose and the true rules will be discussed below.

It is convenient to state the same set of rules in tabular form as given in Table 4.1.

The complete set of rules to be followed for the problem in Figure 4.4 are listed in Table 4.2. The rules are postulated such that the primary body moves exactly opposite to the direction of the secondary body. Thus for example, if the secondary body is seen to lie North-West of the primary body, (as in Figure 4.4) then the *escape direction* of the primary body must be South-East (also shown in Figure 4.4).

In general, the fuzzy rules are applied in the following manner. (See also

| *Rule* | IF *Distance of* S *is* | AND *Angle of* S *w.r.t.* P *is* | THEN *Move* P |
|---|---|---|---|
| *1* | Near | South | North |
| *2* | Near | South-East | North-West |
| *n* | ... | ... | ... |

Table 4.1: Fuzzy rules in tabular form

Section 3.5)

1. Before execution begins, set up the fuzzy controller using the given rule set.
2. During execution, calculate (or directly read in) the estimated values of the input parameters, namely distance and angle.
3. Fuzzify the input parameters using an appropriate bandwidth to yield a fuzzified distance and fuzzified angle.
4. Feed in the fuzzified angle and fuzzified distance into the fuzzy controller.
5. Defuzzify the fuzzy output representing angle to obtain the crisp output value.
6. Interpret the crisp value of output to give a motion command to the primary body.

### 4.3.2 Rotation of frames

Table 4.2 lists a set of rules that includes at least one rule for each direction. Thus there are eight set of rules. Each rule is defined over the Universe $U_D \times U_A \times U_A$ and is stored inside the computer as a matrix of dimension $\mathrm{N}_D \times \mathrm{N}_A \times \mathrm{N}_A$. Storing a set of eight rules in this form consumes a lot of memory which deters us from adding more rules to take care of complex cases. However, it is observed from Table 4.2 that each rule essentially conveys the same idea — that the primary body must

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Distance of* S *is* | *Angle of* S *w.r.t.* P *is* | *Move* **P** |
| *1* | Near | East | West |
| *2* | Near | North-East | South-West |
| *3* | Near | North | South |
| *4* | Near | North-West | South-East |
| *5* | Near | West | East |
| *6* | Near | South-West | North-East |
| *7* | Near | South | North |
| *8* | Near | South-East | North-West |

Table 4.2: A set of rules covering all directions in Figure 4.4

move exactly opposite to the perceived direction of the secondary body. Thus, Table 4.2 carries a lot of redundancy in its statements.

To overcome the disadvantage of repeating the same rules for each direction and to conserve computer memory, the concept of "rotation of frames" is introduced. This concept is explained with the help of Figures 4.5 and 4.6. Figure 4.5 shows the frame $X_0$-$Y_0$ in which the $X_0$-axis is aligned with the *East* direction. In this frame it is noted that the secondary body appears to be situated *North-West* of the primary body. It is desired to rotate the frame $X_0$-$Y_0$ so that the secondary body appears to be located *West* of the primary body. The rotated frame $X_0^R$-$Y_0^R$ is shown in Figure 4.6.

The rotation of frames are mostly done in the fuzzy domain of *Universe of Angle* as defined in Section 3.4.1. This rotation can be achieved by a simple rotation of the fuzzy set representing angle over the Universe $U_A$. It is possible to do the rotation because the Universe of Angle is cyclic in nature. Thus the grade of membership which "falls over" beyond 360°, just "wraps over" to appear at 0°. For instance, we consider rotation of the angle shown as $\theta \approx 135°$ in Figure 4.5.

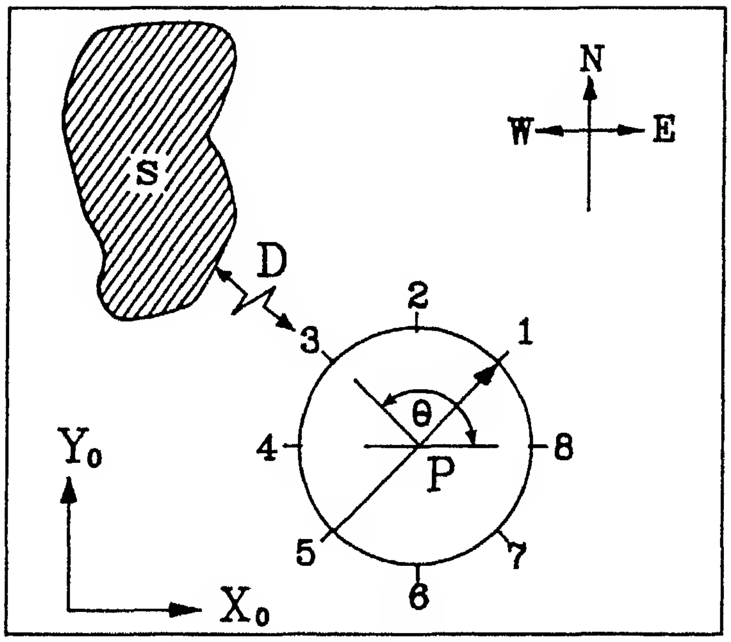


Figure 4.5: Original frame

This angle may be represented in fuzzy notation as,

$$\Theta = \left\{ \frac{0}{0}, \frac{0.1}{45}, \frac{0.75}{90}, \frac{1}{135}, \frac{0.75}{180}, \frac{0.1}{215}, \frac{0}{270}, \frac{0}{315} \right\}.$$

It is observed that a frame rotation of 45° in the clockwise direction is necessary to make the secondary body appear *West* of the primary body. This is achieved by shifting the membership elements of Θ such that the highest membership value appears at the element value '180' in the fuzzy set. Thus the rotated angle in our example (see Figure 4.6) is

$$\Theta^R = \left\{ \frac{0}{0}, \frac{0}{45}, \frac{0.1}{90}, \frac{0.75}{135}, \frac{1}{180}, \frac{0.75}{215}, \frac{0.1}{270}, \frac{0}{315} \right\}$$

A rotation of +45° is equivalent to shifting the elements by 1 unit over the fuzzy angle toward the right. A shift of $p$ units over the Universe of Angle (representing a rotation of $p \times 45°$ in the clockwise direction) is denoted by the operation $Rot^{[p]}$ where $p$ represents the units over which the rotation was affected. The rotation

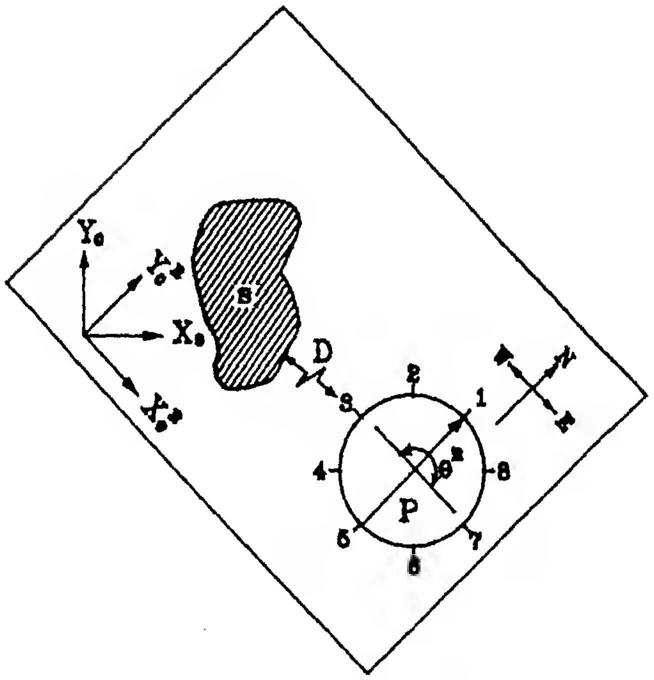


Figure 4.6: Rotated frame

operation can thus be represented in short as

$$\Theta^R = Rot^{[p]}(\Theta) \tag{4.1}$$

It is remarked that since rotations are done in the fuzzy domain, the secondary body will not be situated exactly *West* of the primary body after rotation. In most cases it is likely to lie anywhere from *South-West* to *North-West* of the primary body. This is natural since fuzzy logic is being used to make decisions with approximate data. Since it is known that `Estimated Angle` will be inaccurate, it is illogical to consider it to be an exact crisp number. The fuzzy rules would still trigger because the span of the term *West* also includes portions of *South-West* and *North-West*.

The set of eight rules can now be replaced with just one rule which evokes the required escape strategy.

Suppose the application of this rule results in an escape direction represented

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Distance of* **S** *is* | *Angle of* **S** *w.r.t.* **P** *is* | *Move* **P** |
| *1* | Near | West | East |

Table 4.3: Compact set of rules in rotated frame

as

$$\Theta^R_{esc} = \left\{ \frac{0.5}{0}, \frac{0.14}{45}, \frac{0}{90}, \frac{0}{135}, \frac{0}{180}, \frac{0}{215}, \frac{0}{270}, \frac{0.14}{315} \right\}.$$

The crisp value of this angle may be found out by the centre of gravity method, and it is intuitively obvious that $\theta^R_{esc} \approx 0°$. The actual direction of escape will be given by the relation

$$\Theta_{esc} = Rot^{[-p]}(\Theta^R_{esc}) \tag{4.2}$$

which describes a rotation over $(-p)$ units i.e., an *inverse rotation* over $p$ units. This is essentially a rotation of $p \times 45°$ in the counter-clockwise direction.

The rules expressed in Table 4.3 do not express an intelligent way of escape for the primary body. For instance, when the primary body, **P** is approached by the secondary body **S**, body **P** moves exactly opposite to the location of **S**. Thus it will never be able to move away from the path of the obstacle. An intelligent set of rules should move the primary body cleverly so that it deviates from the path of approach of the secondary body. Table 4.4 tabulates a new set of rules for the same situation that are formed more judiciously. When the object is being attacked exactly from behind, it must deviate from the path to make way for the obstacle. A "preferential" direction of escape is thus assigned for the object when attacked *exactly* from behind. This preferential direction is taken to be *North-East* for the case when S is perceived to be exactly *West* of **P**. The arrows in Figure 4.7 show the possible directions of escape of the primary body when the secondary body is *somewhat* West of the object.

### 4.3.3 Example of two-body interaction

*Example 1 :* The set of rules listed in Table 4.4 were implemented on a computer

| *Rule* | IF<br>*Distance of* S *is* | AND<br>*Angle of* S *w.r.t.* P *is* | THEN<br>*Move* P |
|---|---|---|---|
| *1* | Near | West | North-East |
| *2* | Very Near | West | East |
| *3* | Near | North-West | South-East |
| *4* | Near | North-East | South-West |

Table 4.4: Set of rules incorporating preferential direction

and the results are shown in Figures 4.8 through 4.10. This example highlights the application of fuzzy rules for various combinations of primary and secondary bodies. We are still considering a two-body interaction problem, but more than two bodies are chosen on the same plane to show numerous combinations.

Figure 4.8 shows a set of 5 bodies B0, B1, B2, B3 and B4. Body B0 is considered as the only secondary body while bodies B1, B2, B3 and B4 are considered as primary bodies in turn. Body B0 has started from the bottom of the frame and is proceeding towards the top. The primary bodies are initially static and will move only when they sense an imminent collision. Figure 4.9 shows the same scene at a more advanced stage where the body B0 has already proceeded towards the middle of the frame. By this time bodies B1 and B2 sense collision with B0 and hence move away from its path. Body B1 is observed to move towards the "left" of the path as suggested by the *preferential direction* of the rule set given in Table 4.4. In Figure 4.10 the body B0 has already reached towards the end of its path. The bodies B4 and B3 move away towards their right because of the effect of rule 3 in Table 4.4. However, body B2 does not move beyond its position in Figure 4.9.

It is to be observed that although bodies B2 and B3 need not have moved to avoid collision, they have apparently moved away to a safer position than before. This is because the distances involved in the rules are fuzzy quantities, and the bodies will move according to how the distance "Near" is perceived. This perception is dependent on the *definition* of the fuzzy parameter "Near". In this problem "Near" could have been defined to include a smaller distance, thus avoid-

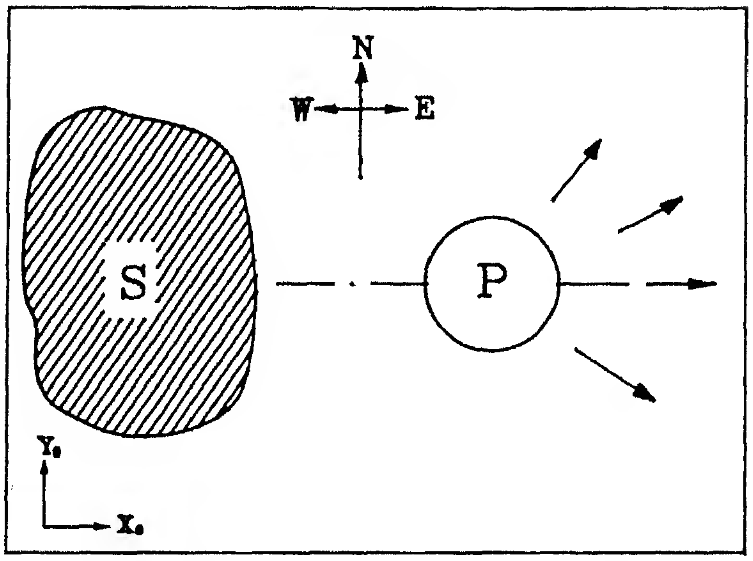


Figure 4.7: Possible escape directions of object

ing unnecessary movements. However, a small definition tends to make the system unreliable. It is emphasized that since we are dealing with fuzzy angles defined over a universe whose gradation is 45°, the set of rules obviously cannot cover the entire range. If the angle happens to lie between two consecutive elements of the universe (which would be around e.g. $22\frac{1}{2}^\circ$, $67\frac{1}{2}^\circ$, $112\frac{1}{2}^\circ$ ,...), then the bodies tend to come too close to each other before getting detected, although the fuzzy compositional rule of inference ensures that all directions are implicitly included if the bandwidth is properly chosen. For such a case, it is always better to keep a *margin of safety* by encompassing a larger distance in our definition of "Near" and by defining a larger bandwidth of angle. A reliable set of rules which works for almost all cases must strike a good compromise between the gradation of the Universe of Angle and the definition of the fuzzy quantities defined over the Universe of Distance. A better reliability analysis of the fuzzy algorithm is given in Section 4.4.6.

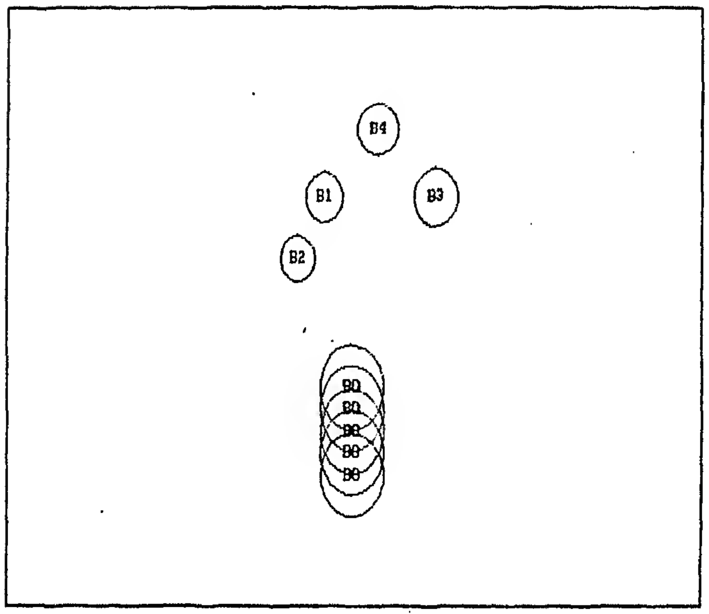


Figure 4.8: Initial position of objects

## 4.4 Multiple Body Interaction

### 4.4.1 Formulation of fuzzy rules

The case dealt in the earlier section involves two bodies, one of which is static unless influenced by another body, and another one moving unconcerned about its environment. A practical situation may demand that bodies avoid collision while simultaneously moving towards their respective goal positions. In the general statement of the problem, as defined in Section 4.2, all bodies have a known starting position and a known goal position (or positions). Each body avoids collision by sensing the presence of other bodies. This section will address the problem of multiple body collision avoidance, all of which are moving towards their respective goals.

The choice of a good starting path is impossible in the absence of any prior knowledge about the environment. Thus all bodies are initially directed to approach the current goal positions along a straight line, i.e. the path which leads

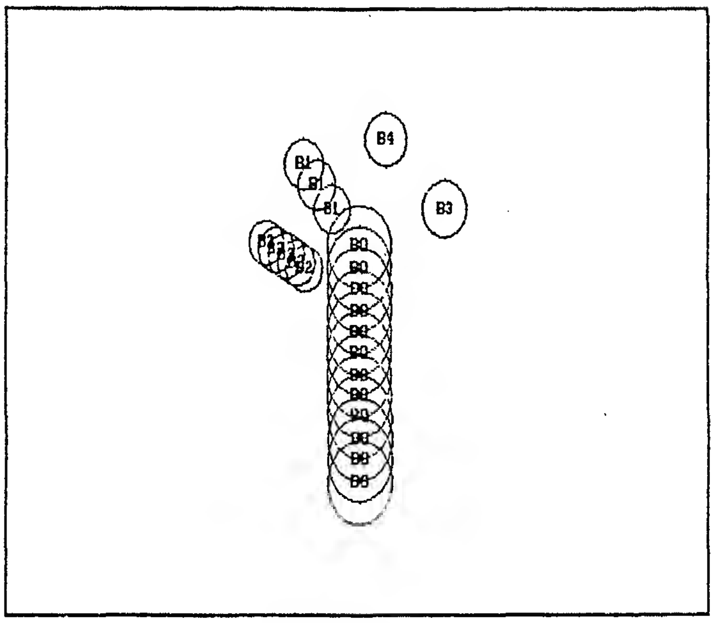


Figure 4.9: Position showing escape of two bodies

to the goal in the shortest time. When an external body is sensed nearby, the bodies are allowed to adopt an avoidance measure by deviating from the assigned path. When the danger of collision is perceived to be over, the bodies are made to resume their journey toward their goals. The amount of deviation necessary to avoid an impact, is decided *individually* by the controller of each body by basing its opinion on locally perceived instantaneous information.

To study the interaction between multiple bodies, it is essential to look at each *pair* of bodies in isolation. While developing rules for the problem in Section 4.3.1 an important advantage that allowed us to ignore the *velocity* of the bodies was the fact, that the primary body was static and did not have a goal. In the current problem, since both the primary and secondary body have a definite motion, this problem must include velocity terms in the rule set. The earlier convention of seeing the secondary body *West* of the primary body by an appropriate rotation of frames is maintained. Let us denote the velocity of the primary body as $\vec{v}_P$ and that of the secondary body as $\vec{v}_S$. Figure 4.11 lists a few cases where the

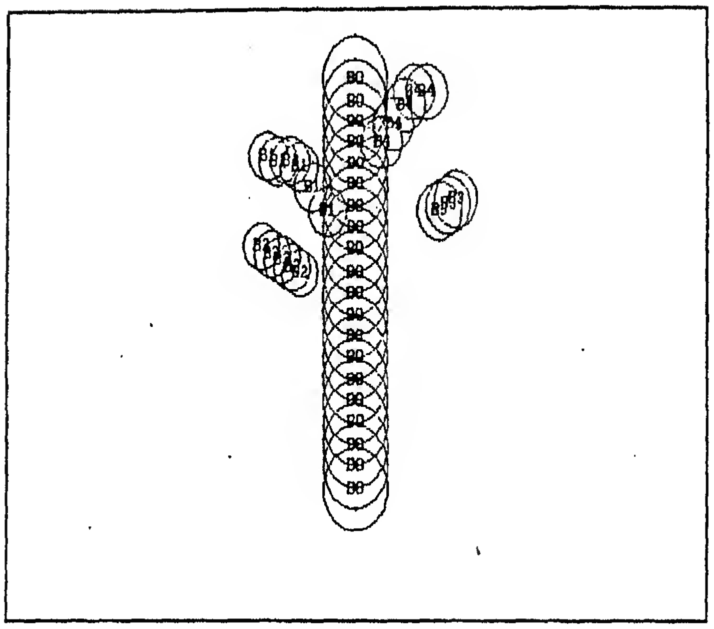


Figure 4.10: Final position attained by the bodies after execution

velocities of the bodies are shown with arrows beside the respective bodies. The lower column is the velocity diagram of the two bodies. $\vec{v}_{SP}$ represents the relative velocity of S with respect to P. Thus $\vec{v}_{SP}$ may be represented as

$$\vec{v}_{SP} = \vec{v}_S - \vec{v}_P. \tag{4.3}$$

In all figures the separation between the bodies has been kept constant to highlight the influence of the parameter $\vec{v}_{SP}$ upon the escape direction. A careful examination of the cases shown in Figure 4.11 reveals that a collision between the two bodies is imminent only when the relative velocity vector points towards *East.* The acuteness of collision is dependent directly on the angle the vector $\vec{v}_{SP}$ subtends from East — the smaller the angle, more the chance of collision.

A set of rules are presented that will deviate the primary body with respect to the secondary when it perceives the relative velocity vector to be *East* (or near about) and distances to be *Near.* In this problem the preferential direction of escape has been set as *South-East.* The resulting effect is that when two bodies

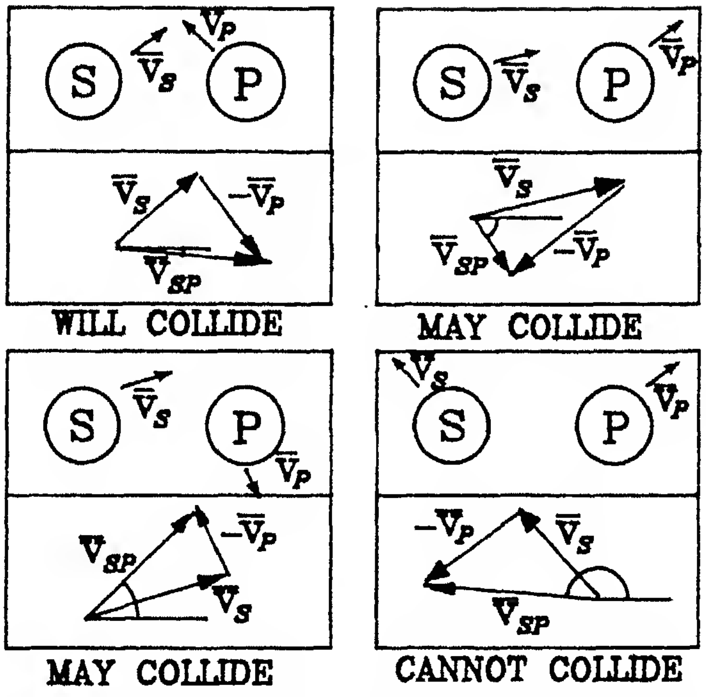


Figure 4.11: Relative velocity of the secondary body for different cases

appear to collide head-on, both bodies deviate towards the *left* of their line of approach. Table 4.5 lists the new set of rules for this problem.

The above set of rules makes use of the relative velocity of S with respect to P. Since sensors will only measure the distance vector, velocity can be found by observing distance over a period of time. The sampling time is considered as $\Delta t$ and distances are measured at intervals of $\Delta t$.

Figure 4.12 explains how the relative velocity of the secondary body is assessed by observing its distance from the primary body over a period of time. Let $S_0$ and $P_0$ represent the position of the secondary body and the primary body respectively at an instant of time, say $t_0$. At this instant, the primary body observes the secondary body to be at a distance $d_0$ from P. The position vector of S with respect to P at this instant is represented as $\vec{d_0}$ in Figure 4.12. At the next instant of sampling, $t_1 = (t_0 + \Delta t)$, the position of the secondary body and the primary body is represented as $S_1$ and $P_1$ as shown in Figure 4.12. Thus the velocity

| *Rule* | IF *Distance between* S *and* **P** *is* | AND *Rel. Velocity Direction of* S *w.r.t.* **P** *is* | THEN *Escape Velocity Direction of* **P** *w.r.t.* $\vec{v}_P$ *is* |
|---|---|---|---|
| *1* | Very Near | North-East | South-East |
| *2* | Very Near | South-East | North-East |
| *3* | Quite Near | East | North-East |

Table 4.5: Set of rules for a moving pair of bodies

vector of **P**, $\vec{v}_P$ is the vector $\overline{\mathbf{P}_1\mathbf{P}_0}$. Similarly, $\vec{v}_S = \overline{\mathbf{S}_1\mathbf{S}_0}$. Let $\vec{d}_1$ be the perceived distance vector of S with respect **P** at this instant. Then the relative velocity of S with respect to **P**, $\vec{v}_{SP}$ can be obtained in terms of distances from

$$\vec{v}_{SP} = \frac{\vec{d}_1 - \vec{d}_0}{\Delta t} \tag{4.4}$$

as can be easily deduced from Figure 4.12.

### 4.4.2 Input parameters to fuzzy controller

#### 1. Distance based on predicted position of bodies

The fuzzy rules are applied in accordance with the guidelines developed for two-body interaction in page 73. However, another concept that takes into consideration the previous *history* of the bodies is now introduced into this problem. This concept is described in the next few paragraphs.

The collision avoidance strategy described so far applies to the current position of the objects. However, in tight situations, when the motion of a neighbouring object is liable to change abruptly, it is always safer to keep a safe limit. One way of keeping a safe limit, as described in Section 4.3.1, is to define a "margin of safety". In other words, physically speaking, a *margin of safety* extends the body beyond its boundary to cover a larger area around itself. This definition of margin of safety is quite restrictive because it unnecessarily encloses an area irrespective of the seriousness of danger in that region. There is another way in which a safe

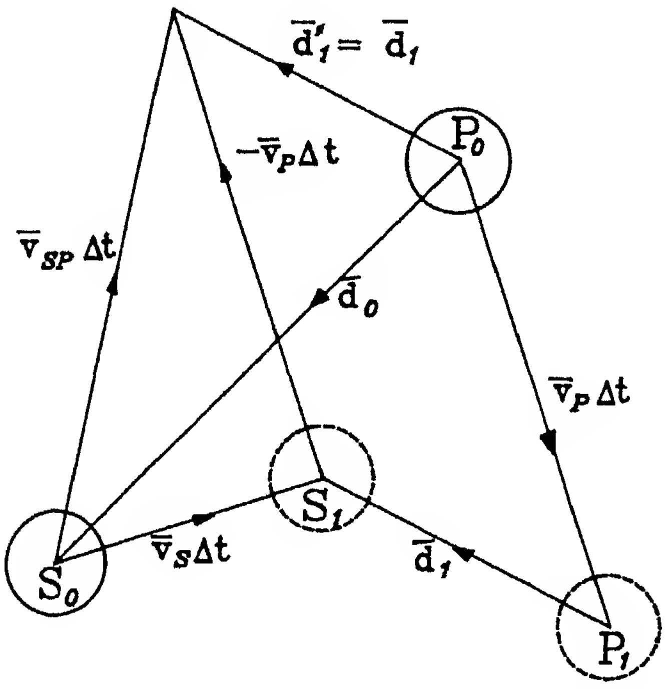


Figure 4.12: Relative velocity from distance measurements

margin can be defined more intelligently. An area is considered to be dangerous only if the primary body is approaching towards that region, and a secondary body is liable to occupy the same region in the near future. This leads us to the concept of "predicted position" of bodies.

The predicted position is simply the linearly extrapolated position of the body from its immediately preceding past position. The resulting position is a *predicted image* of the body. The position occupied by the image will be the actual position if the body decides to stay on the same course without change in velocity. The level of prediction may be set according to a parameter depending on the safety required. This parameter is denoted by $\ell$. Let the past position of a secondary body be $\vec{p}_S^{past}$ and the present position be $\vec{p}_S^{present}$. Then the future image, $\vec{p}_S^{future}$ is calculated from

$$\vec{p}_S^{future} = (\ell + 1)\vec{p}_S^{present} - \ell\vec{p}_S^{past} \tag{4.5}$$

where $\ell$ is the level of prediction, which is normally set to 1. A large level of prediction is inadvisable because the motion of the secondary body is unknown and there are always chances that it might be erratic. This may lead to a large anomaly in the predicted image of the body and the position actually occupied by the body at a future time. Instead of applying the fuzzy rules to the actual position of the secondary bodies, they are now applied to the predicted images of the bodies.

**2. Angle based on relative velocity direction**

The second input parameter in this problem is the relative velocity of approach of S. The fuzzification of the relative velocity vector $\vec{v}_{SP}$ is done on the basis of the formula

$$\mathrm{V}_{SP} = \bigcup_{x=0}^{x=360} \frac{\| \vec{v}^{R}_{SP} \|}{\| \vec{v}^{*} \|} G(x, \theta_{SP} - 60, \theta_{SP} - 20, \theta_{SP}, \theta_{SP}, \theta_{SP} + 20, \theta_{SP} + 60) \quad (4.6)$$

where $\vec{v}^{*}$ is the nominal velocity assigned to all bodies when they are free to move without any possibility of collision and $\theta_{SP}$ is the direction of the vector $\vec{v}_{SP}$.

The escape vector found from the fuzzy rules gives the suggested direction of movement of the primary body under the influence of the secondary bodies. The escape vector may be loosely interpreted as a repulsive force acting away from the nearby bodies. There is also a force of attraction which pulls the primary body towards its current goal position. This vector is called the *attraction vector*. The net motion of the primary body is decided by taking the resultant of the attraction vector and the escape vector.

### 4.4.3 Fuzzy algorithm for multiple body interaction

The sequence of steps to avoid collision for each body is given by the algorithm below.

```
procedure FindEscapeVector(i :  integer);
begin
   for each body j := 1 to n do
     escape_vector := 0;
```

```
    begin
      if (j = i) continue;
      assess the position of body j;
      find the relative velocity of approach by considering the
                past position of body j;
      find the predicted image of body j based on the past
                and present position;
      rotate the frame appropriately;
      fuzzify the input parameters, 'predicted distance' and
                'angle of approach';
      feed in the fuzzy values into the fuzzy controller;
      perform an inverse rotation to get fuzzy vector in
                original frame;
      defuzzify the output parameter to get the escape
                vector pertaining to body j (escape[j]);
      escape_vector = escape_vector + escape[j];
    end;
end;
```

The output of the algorithm according to the rules of Table 4.5 is the relative escape motion of the primary body, **P** in the rotated frame under the influence of the secondary body S. Let us denote the defuzzified output quantity (according to (2.30)[1]) as $\vec{v}^{R}_{P,\,relEsc,\,S}$. In other words, this quantity represents the escape vector of **P** assuming that it was stationary. The actual escape velocity in the rotated frame due to S is then given by

$$\vec{v}^{R}_{P,\,Esc,\,S} = \vec{v}^{R}_{P,\,relEsc,\,S} + \vec{v}^{R}_{P} \tag{4.7}$$

The absolute velocity with respect to the plane coordinate system $X_0$-$Y_0$ is found according to the method outlined in Eq. 4.2.

$$\vec{v}_{P,\,Esc,\,S} = Rot^{[-p]}(\vec{v}^{R}_{P,\,Esc,\,S})$$

[1]Also see Section 3.5.

where $p$ is the number of units by which the original direction vector $\vec{d_1}$ of Figure 4.12 was rotated to make the secondary body appear *West* of the primary body.

The net escape velocity $\vec{v}_{P,\,Esc}$, will be the resultant of the influence of all secondary bodies on the primary body. Thus

$$\vec{v}_{P,\,Esc} = \sum_{i=1,\,\mathrm{S}_i \neq \mathrm{P}}^{n} \vec{v}_{P,\,Esc,\,S_i} \tag{4.8}$$

### 4.4.4 Complexity of the collision avoidance algorithm

For simulation runs, each body is considered as a primary body and the escape vector resulting from the presence of the neighbouring bodies is found. The operations in the crisp domain including assessing the position, finding relative velocity of approach and predicting the future position are linear operations. Thus these operations are of complexity $\mathcal{O}(n)$ where $n$ is the number of bodies. The compositional rule of inference involves ($\mathrm{N}_d\ \mathrm{N}_a^2$) comparisons for every rule that is processed (This is true only for the formulation strategy presented in this chapter, not in general). The overall complexity of finding the escape vector for one body is thus $\mathcal{O}(n) + r_n(\mathrm{N}_d\ \mathrm{N}_a^2)$ where $r_n$ is the number of rules in the fuzzy controller. For simulation of the program, this operation has to be performed for each body.

### 4.4.5 Example to illustrate multiple body interaction

The algorithm described above has been applied to five bodies moving simultaneously on a plane. The starting position of the bodies is shown as $B_1^S$, $B_2^S$, $B_3^S$, $B_4^S$ and $B_5^S$ in Figure 4.13. The goal position of the bodies is depicted as $B_1^G$, $B_2^G$, $B_3^G$, $B_4^G$ and $B_5^G$ in the same figure. The positions attained by the bodies at various stages of execution is given in Figures 4.14 to 4.17. The path traced by all bodies is superposed on all these figures for comparison. It is observed that a collision-free configuration is attained by all bodies during the entire execution.

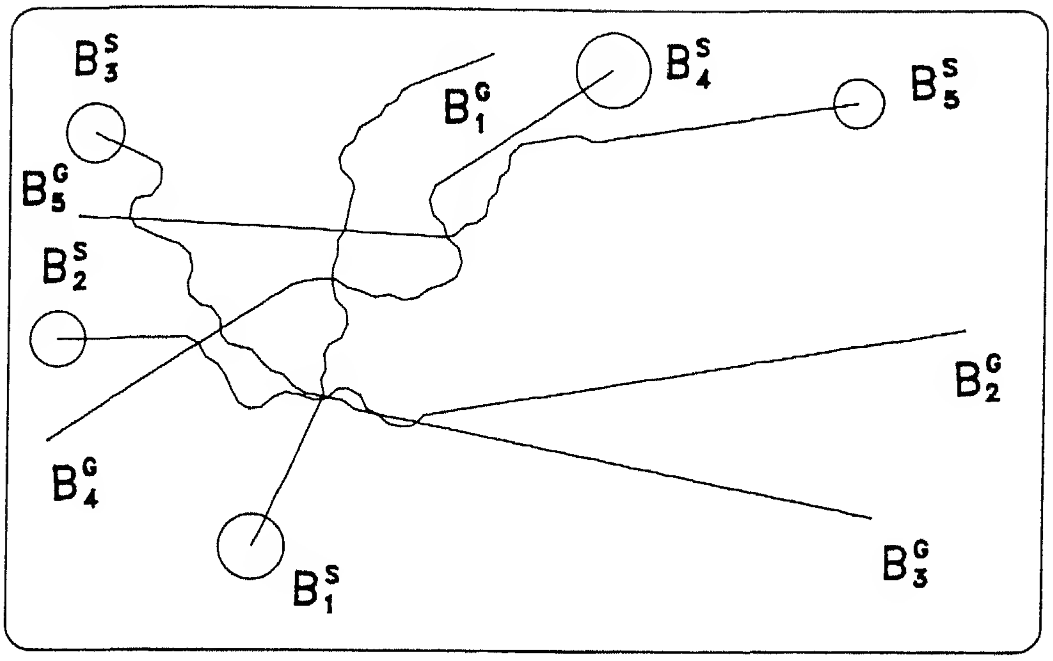


Figure 4.13: Starting configuration of five bodies

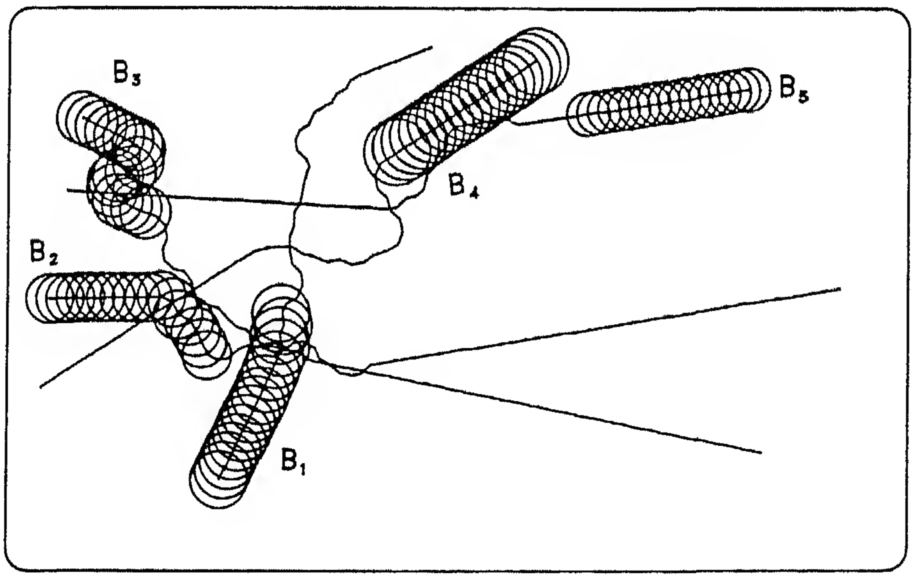


Figure 4.14: Configurations showing collision avoidance maneuvers adopted by bodies : Stage 1

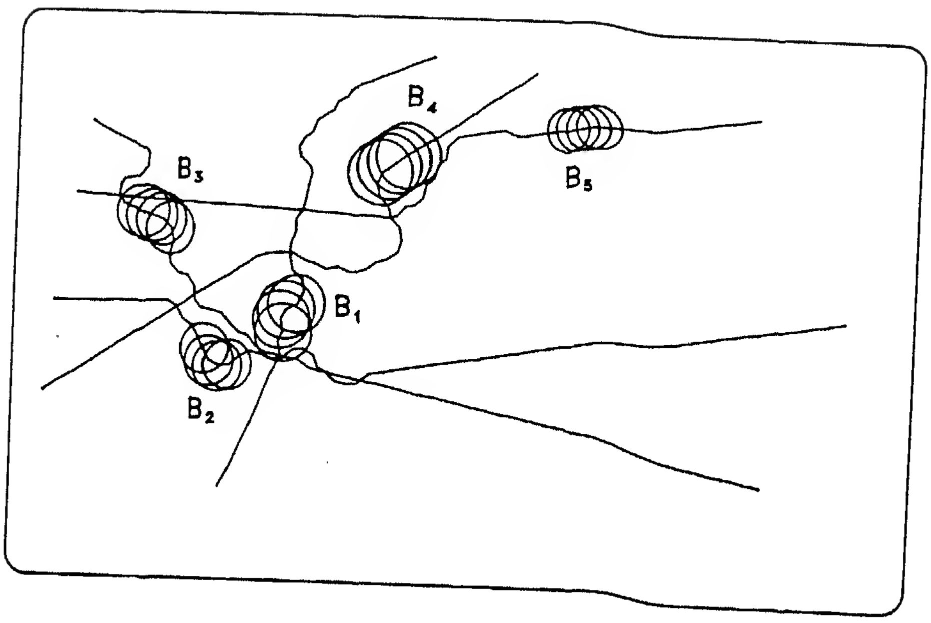


Figure 4.15: Configurations showing collision avoidance maneuvers adopted by bodies : Stage 2

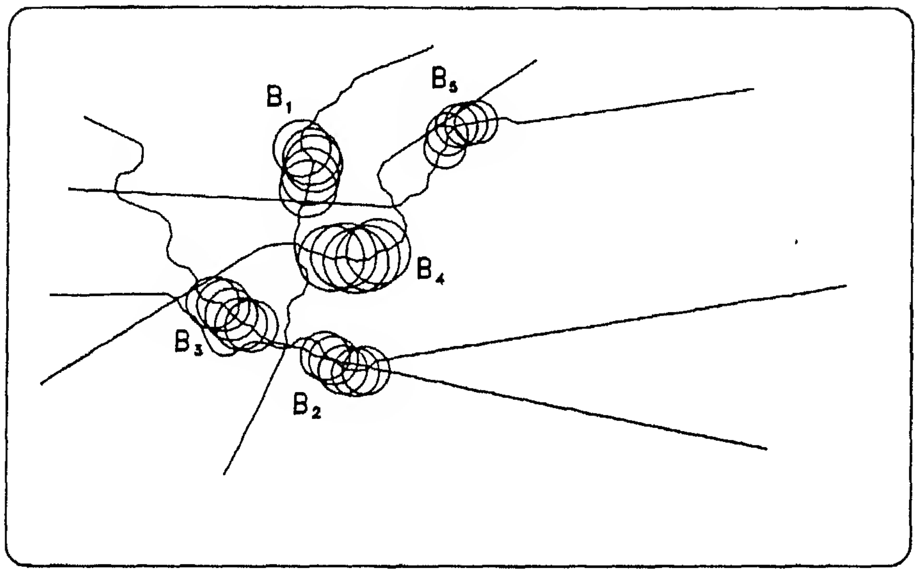


Figure 4.16: Configurations showing collision avoidance maneuvers adopted by bodies : Stage 3

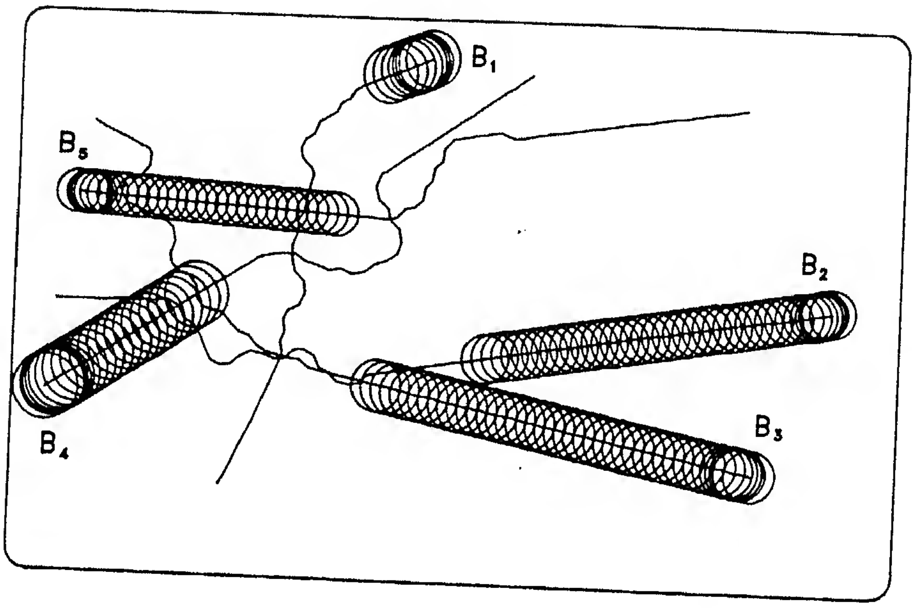


Figure 4.17: Configurations showing collision avoidance maneuvers adopted by bodies : Stage 4

### 4.4.6 Investigating effects of variation of fuzzy control parameters

In this section the effect of various control parameters on the reliability of the fuzzy algorithm is investigated. A fuzzy algorithm has an inherent aspect of unreliability associated with it, and so it cannot be guaranteed to be perfectly fool proof. This is so, because the modelling of a system through a fuzzy controller ignores its mathematical description; rather the modelling is an attempt to emulate the behaviour of the system at the gross level. This leads to a somewhat superficial description of the system, but is helpful to model systems which are either too complex to enable its strict mathematical definition or which have a large number of input parameters. Fuzzy algorithms will always lead to "practical" solutions of complex problems. The algorithm developed in this work can be made "practically" fool proof by introducing a margin of safety and by appropriately defining the fuzzy quantifiers.

A simple example is considered in this section to carry out the failure analysis of the fuzzy algorithm. The primary control parameters that affect the behaviour of the fuzzy algorithm are identified as follows:

1. Gradation of the Universe of Distance.
2. Gradation of the Universe of Angle.
3. Meaning of the Fuzzy terms over Distance, especially "Near", used for defining fuzzy rules.
4. Meaning of the Fuzzy terms over Angle used for defining fuzzy rules.
5. Bandwidth used over Distance for fuzzifying the input parameter.
6. Bandwidth used over Angle for fuzzifying the input parameter.
7. The total number of rules used in the fuzzy controller and the statements of each rule.

Among the seven factors listed above, the gradation of the Universe of Distance and Angle (i.e., the number of elements in each universe) has not been found to affect results too much. The effect of the latter five factors leads to interesting inferences.

First, the effect of changing the *meaning of the term "Near"* and the *number of rules* is investigated. These items are listed as 3 and 7 in the above list. An example shown in Figure 4.18 is considered in which a body $B_2$ is to start from $B_2^S$ and reach its goal $B_2^G$ while avoiding the static body $B_1$. The problem is first solved with one rule set given in Table 4.6, and the meaning of "Near" assigned as

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Distance between* S *and* **P** *is* | *Rel. Velocity Direction of* S *w.r.t.* **P** *is* | *Escape Velocity Direction of* **P** *w.r.t.* $\vec{v}_P$ *is* |
| *1* | Near | East | South |

Table 4.6: One rule for Figure 4.18

$$M(Near) = \bigcup_{x \in \mathbf{U}_D} G(x,0,0,0,0,25,50). \tag{4.9}$$

It is observed from the figure that the bodies intercept proving thereby that the choice of parameters is not correct. It is thus decided to change the meaning of "Near" to

$$M(Near) = \bigcup_{x \in \mathbf{U}_D} G(x,0,0,0,0,50,100). \tag{4.10}$$

keeping the same rule. The new trajectory of body $B_2$ is shown in Figure 4.19. It is observed that the bodies still intercept, though by a lesser amount. It is easy to note from this figure that body $B_2$ had started deviating from the path earlier than it did in Figure 4.18 because of the somewhat *safer* definition of "Near". But one rule is still inadequate to prevent a collision at a later stage of the execution because this rule does not include the possibility of $\vec{v}_{SP}$ being perceived in *North-East* direction in the rotated frame. Thus it is decided to increase the number of rules to two. The new set of rules is given in Table 4.7. The same problem is solved with these two rules and "Near" defined as in (4.10). The resulting trajectory of

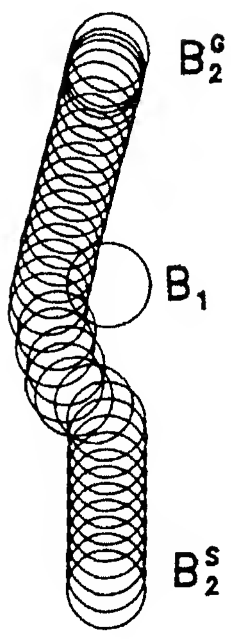


Figure 4.18: Motion of body $B_2$ with one rule set and "Near" defined from 0 to 50

$B_2$ is shown in Figure 4.20. The bodies are observed to avoid collision with a *large* margin of safety which might not always be desirable from the point of view of efficiency. It thus leads us to reconsider our definition of "Near" and revert it back to its old value as defined in (4.9). The same problem is solved again and the results appear in Figure 4.21. This figure presents a satisfactory picture of the collision avoidance strategy. A plot of the 'distance of separation' between the bodies $B_1$ and $B_2$, with respect to 'execution steps' for the four examples presented in the preceding paragraphs is given in Figure 4.22. The horizontal dashed line appearing in the graph marks the zero level. A plot which goes below this level indicates a case of collision. The best plot must appear marginally above the zero level of separation when the two bodies touch each other. Among the four cases depicted in the figure, the lower two curves should be rejected because they indicate a case of collision. These two curves pertain to the first two problems presented in this section. The topmost curve shows a large margin of safety and is thus not a useful candidate, although it indicates a collision free path. The second curve from the

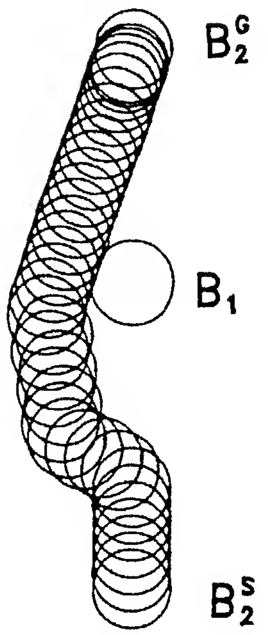


Figure 4.19: Motion of body $B_2$ with one rule set and "Near" defined from 0 to 100

top is best among all curves shown in this figure and pertains to the last example in this section.

The effect of changing the definition of the term "Near" is depicted in Figure 4.23. In this figure the X-axis represents the value of $a_2$ (as defined in (2.24)) chosen for defining the term "Near". The Y-axis represents the minimum separation attained by the two bodies during the run. It is observed that when the value of $a_2$ is below 40, collision cases occur. However a value of $a_2 \geq 45$ is observed to be safe for the example considered herein.

The effect of changing the band-width used for defining the meaning of the terms "East", "North-East" etc. upon the 'minimum separation' is depicted in Figure 4.24. In this figure the X-axis represents the value of the half-bandwidth $= (a_2 - a_1)/2$ (as defined in (2.24)) chosen for defining the angles in the fuzzy rules. The Y-axis again represents the minimum separation attained by the two bodies during the run. It is observed that when the value of half-bandwidth is below 50,

| *Rule* | IF<br>*Distance between*<br>S *and* P *is* | AND<br>*Rel. Velocity Direction*<br>*of* S *w.r.t.* P *is* | THEN<br>*Escape Velocity Direction*<br>*of* P *w.r.t.* $\vec{v}_P$ *is* |
|---|---|---|---|
| *1* | Near | East | South |
| *2* | Near | North-East | South |

Table 4.7: Two rules for Figure 4.20

collision cases occur. However a value of half-bandwidth $\geq 50$ is observed to be safe.

The effect of changing the *bandwidth of distance* is investigated now. This appears as item 5 in the list at the beginning of this section. A plot of the 'minimum separation' versus the 'bandwidth of distance' appears in Figure 4.25. Collision cases are found to disappear beyond a value of 5, while for values greater than 10, the minimum separation is not affected by any further increase in the bandwidth value.

The effect of changing *bandwidth of angle* (item 6) is now investigated. A plot for 'minimum separation' versus 'bandwidth of angle' is given in Figure 4.26. It is observed that beyond a value of 25°, the minimum separation is not affected by any change in the bandwidth.

It is thus concluded that most of the control parameters used in the fuzzy algorithm need to be *above* a certain optimum level for best performance. It is also observed that assigning a very high value does not lead to any further improvement.

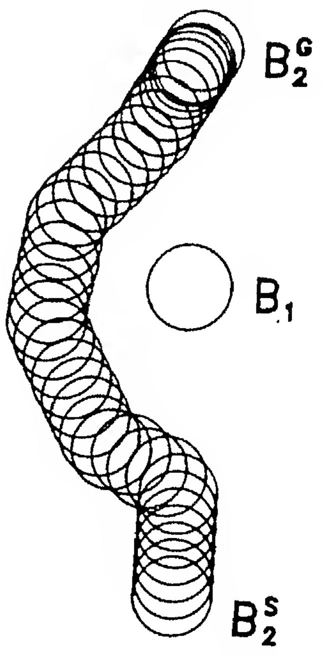


Figure 4.20: Motion of body $B_2$ with two rules and "Near" defined from 0 to 100

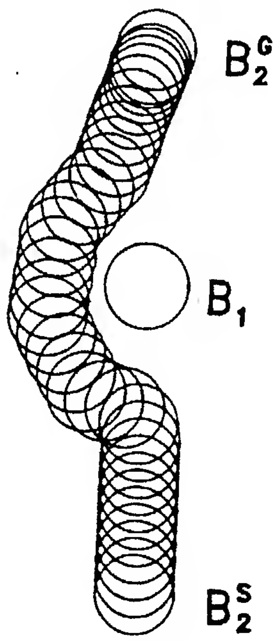


Figure 4.21: Motion of body $B_2$ with two rules and "Near" defined from 0 to 50

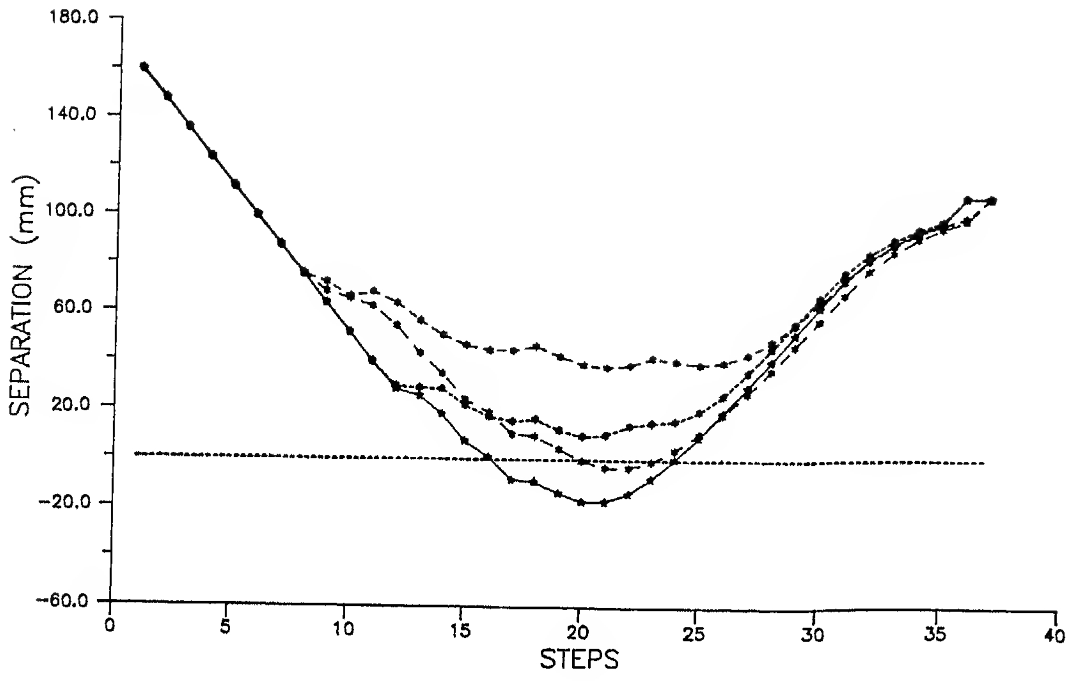


Figure 4.22: Plot of 'Distance of separation' versus 'Execution steps' for Figures 4.18 to 4.21

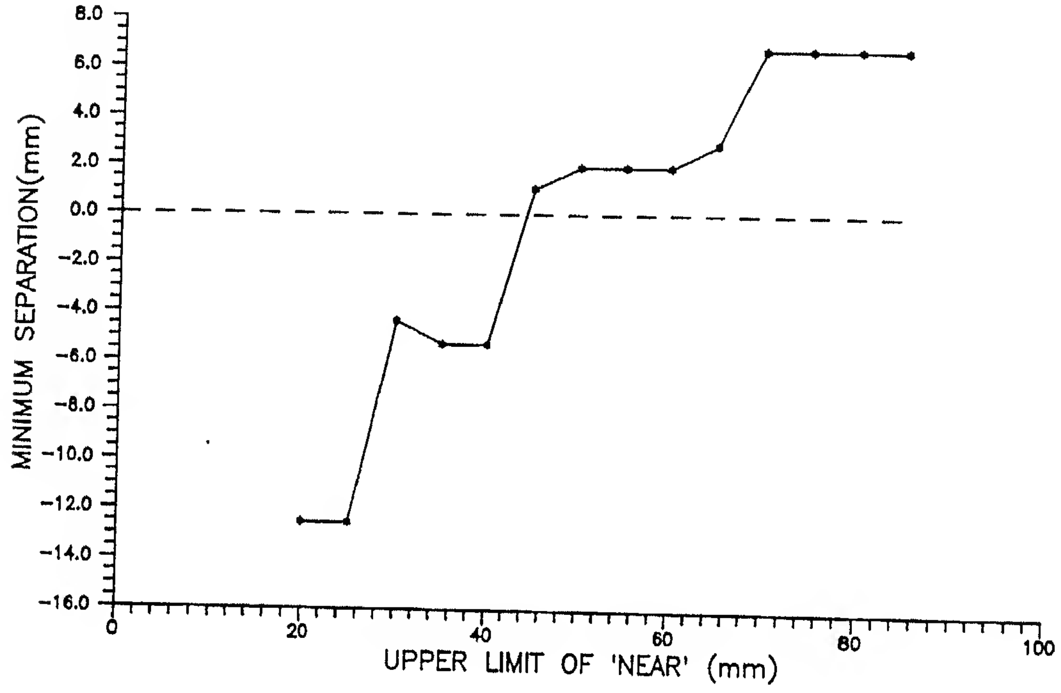


Figure 4.23: Plot of minimum separation versus value of $a_2$ used for defining "Near"

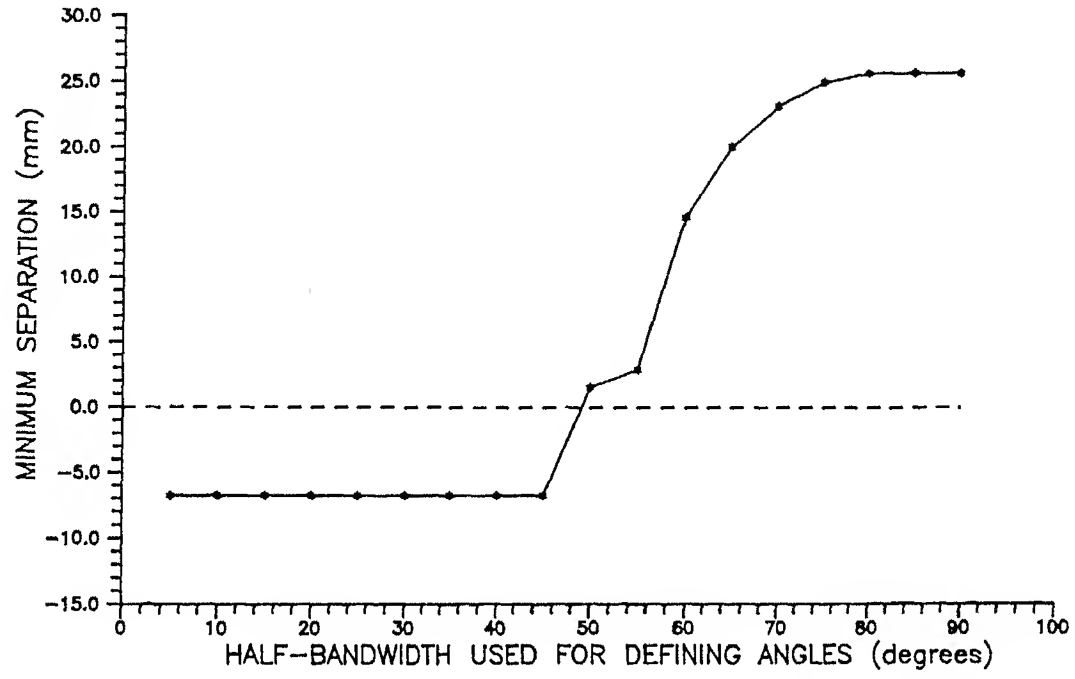


Figure 4.24: Plot of minimum separation versus value of bandwidth used for defining "East", "North-East" etc.

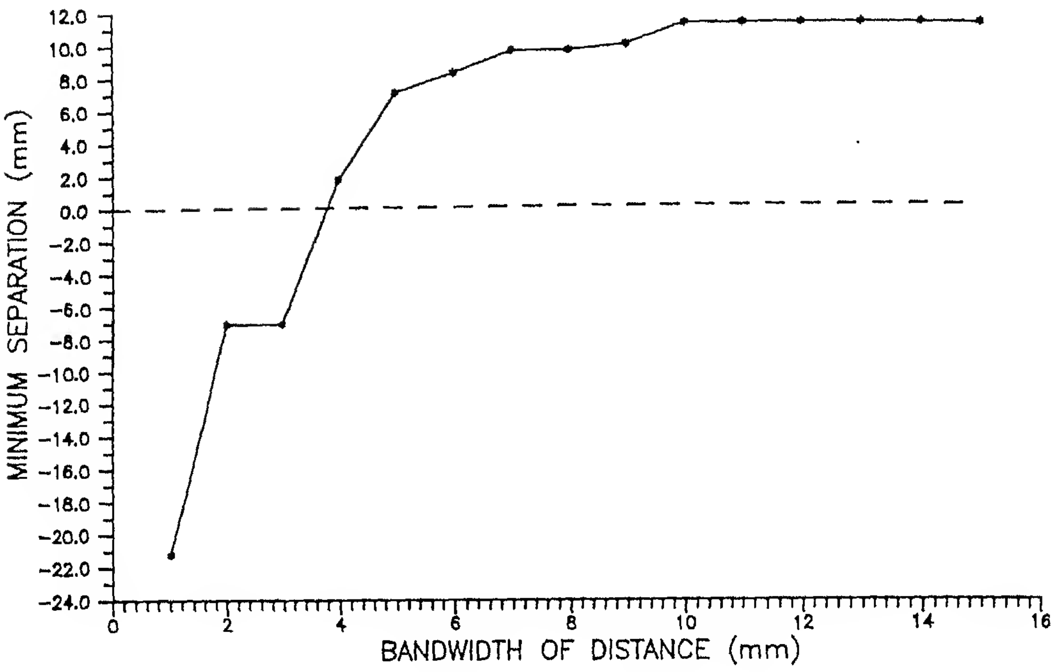


Figure 4.25: Plot of minimum separation versus value of bandwidth of distance

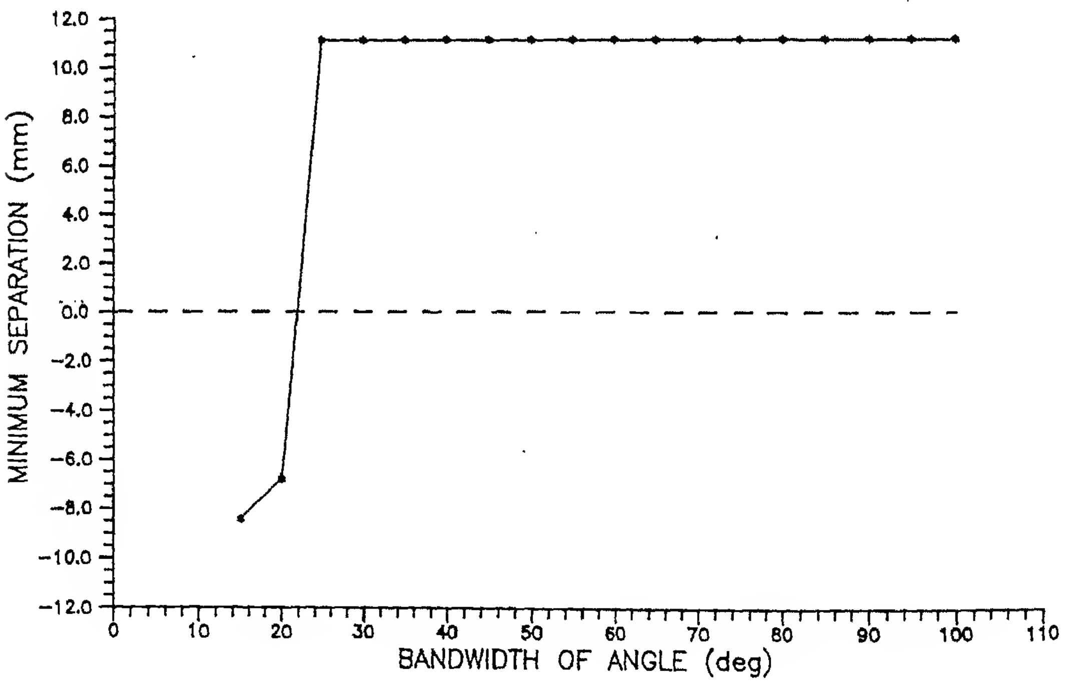


Figure 4.26: Plot of minimum separation versus value of bandwidth of angle

## 4.5 Constraints on Motion Parameters

This section proposes some methods of imparting motion to the bodies satisfying some kinematic constraints. Application of fuzzy rules might yield an escape vector that could be impossible to attain practically because of the practical limitations of the vehicle. Specifically, we wish to impose constraints on the maximum speed, maximum acceleration (or deceleration) and rate of turning of the bodies on the plane. Path constraints are imposed by defining two kinds of bodies viz., *path-preference* and *point-preference* bodies as mentioned in Section 4.2. All constraints are purely geometric and remedial steps to overcome the limitations are also based on geometry.

### 4.5.1 Speed and acceleration limits

The velocity of escape of the primary body $\vec{v}_{P,Esc}$ as obtained by the method outlined in Section 4.4, might sometimes be infeasible to attain because of too high values. This section outlines a method to impart motion to the bodies, keeping in view the velocity and acceleration limitations of the body. Remedial measures to be taken when the desired resulting motion overshoots these limits are also described.

A simplified notation is now introduced to explain the kinematic constraints explained in this section. Let the current velocity of the primary body **P** be represented as $\vec{v}_{P,i}$. The suggested final velocity of **P** (according to the fuzzy rules) is $\vec{v}_{P,Esc}$. The final velocity imparted to **P** after imposing the kinematic constraints will be represented as $\vec{v}_{P,f}$. The acceleration of **P** will be represented as $\vec{a}_P$ which has two components $(a_P)_t$ and $(a_P)_n$, the former along the tangent to the path and the latter along the normal to the path. The instantaneous velocity vector $\vec{v}_P$ always points in the direction of the front vector $\hat{f}$. The angular velocity of turning the front vector $\hat{f}$ is represented by $\omega$. The magnitude of a vector $\vec{v}$ will be denoted as $\|\vec{v}\|$ and its direction as $Dir(\vec{v})$.

Only three kinds of kinematic limits to each body are imposed.

**Constraint 1** The velocity limit which requires that $\|\vec{v}_P\| \leq v_{\max}$.

**Constraint 2** The acceleration limit which requires that the instantaneous tangential acceleration $\vec{a}_P$ should satisfy $(\vec{a}_P)_t \leq a_{\max}$.

**Constraint 3** The angular velocity limit which requires that $||\omega|| \leq \omega_{\max}$.

A feasible solution must **simultaneously satisfy all the above conditions.**

The three cases are described individually and measures to be adopted when limits are exceeded are suggested. These cases are described with the aid of Figures 4.27 and 4.28.

### Keeping velocity limits

Figure 4.27 shows a case where the maximum velocity limit has been exceeded, i.e., $||\vec{v}_{P,\,Esc}|| > v_{\max}$. The remedy is obviously to assign

$$\begin{aligned} ||\vec{v}_{P,\,f}|| &= v_{\max} \\ Dir(\vec{v}_{P,\,f}) &= Dir(\vec{v}_{P,\,Esc}) \end{aligned}$$

The resulting vector $\vec{v}_{P,\,f}$ is shown in Figure 4.27.

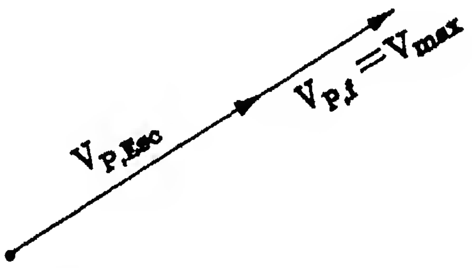


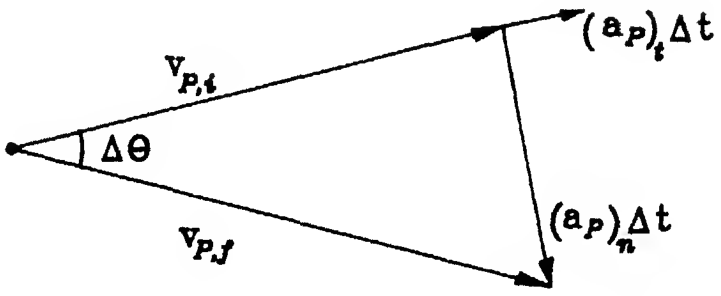
Figure 4.27: Keeping velocity limits

### Keeping acceleration limits

The normal and tangential components of the acceleration $\vec{a}_P$ may be represented as (see Figure 4.28)

$$\begin{aligned} (\vec{a}_P)_t &= \lim_{\Delta t \to 0} \frac{\Delta \vec{v}_P}{\Delta t} = \dot{v}_P \\ (\vec{a}_P)_n &= \lim_{\Delta t \to 0} \vec{v}_P \frac{\Delta \theta}{\Delta t} = ||\vec{v}_P||\omega \end{aligned}$$

It is desired to impose a restriction on the tangential value of the acceleration vector and the value of $\omega$.



$$||(a_P)_t||\Delta t = ||\mathbf{v}_{P,f}|| - ||\mathbf{v}_{P,i}||$$
$$||(a_P)_n||\Delta t = ||\mathbf{v}_{P,i}||\,\Delta\theta$$

Figure 4.28: Keeping acceleration limits

If the tangential acceleration is found to exceed the maximum value, i.e. $(\vec{a}_P)_t > a_{\max}$, then the final value of the velocity is found from

$$||\vec{v}_{P,\,f}|| = sign(||\vec{v}_{P,\,f}|| - ||\vec{v}_{P,\,i}||) \cdot a_{\max}\Delta t + ||\vec{v}_{P,\,i}||$$

where the *sign* function is defined as

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{4.11}$$

The turning rate is found from

$$\omega = \frac{\vec{v}_{P,\,i} \cdot \vec{v}_{P,\,f}}{||\vec{v}_{P,\,i}||\;||\vec{v}_{P,\,f}||}/\Delta t \tag{4.12}$$

When the value of $\omega$ overshoots its limit, $\omega = \omega_{\max}$ is assigned and $\vec{v}_{P,\,f}$ is recalculated.

### 4.5.2 Path priorities on objects

As described in Section 4.2, there are two kinds of bodies based on the restrictions put on their path of motion — *path-preference bodies* and *point-preference* bodies. Objects of the first kind have a prescribed path while the latter ones do not have a prescribed path, but they need to visit some intermediate points $B_i^{G1}, B_i^{G2}, \ldots, B_i^{Gp}$ while they are on their way to the final goal point $B_i^G$. The following paragraph enumerates the options available for each type of body. classes. It is assumed that the body is moving at the nominal velocity $\vec{v}^*$. The maximum attainable velocity of the body is $\vec{v}_{\text{max}}$.

1. Path-Preference bodies.
   (a) The velocity of the object cannot be altered, i.e. $||\vec{v}|| = \vec{v}^* =$ constant.
   (b) The velocity of the object can be altered.
      i. A deceleration up to speed zero is allowed but it cannot accelerate beyond $\vec{v}^*$ i.e., $0 \leq ||\vec{v}|| \leq v^*$.
      ii. A deceleration up to speed zero and an acceleration up to $\vec{v}_{\text{max}}$ is allowed i.e., $0 \leq ||\vec{v}|| \leq v_{\text{max}}$.
      iii. The body is allowed to go backward on the path as long as it does not exceed the maximum velocity limit i.e., $-v_{\text{max}} \leq ||\vec{v}|| \leq v_{\text{max}}$.
2. Point-preference bodies. The object heads towards the goal directly. When the object is dislodged from its path, it again approaches its target directly from its current position.

First path-preference bodies are considered. These objects are bound to stay on the prescribed path, but their velocity on the path may be altered to avoid collision. Let us represent the assigned path for body $B_i$ by a continuous curve from the start $B_i^S$ to the goal $B_i^G$, as shown in Figure 4.29. Two unit vectors $\hat{n}$ and $\hat{t}$ can be defined such that $\hat{n}$ points towards the instantaneous normal to the path and $\hat{t}$ points towards the instantaneous tangent to the path. The normal component of velocity $(v_{B_i})_n$ for path-preference bodies is zero. The motion constraints of such bodies require that $(v_{B_i})_n$ be set to zero. Hence the motion of the body will be

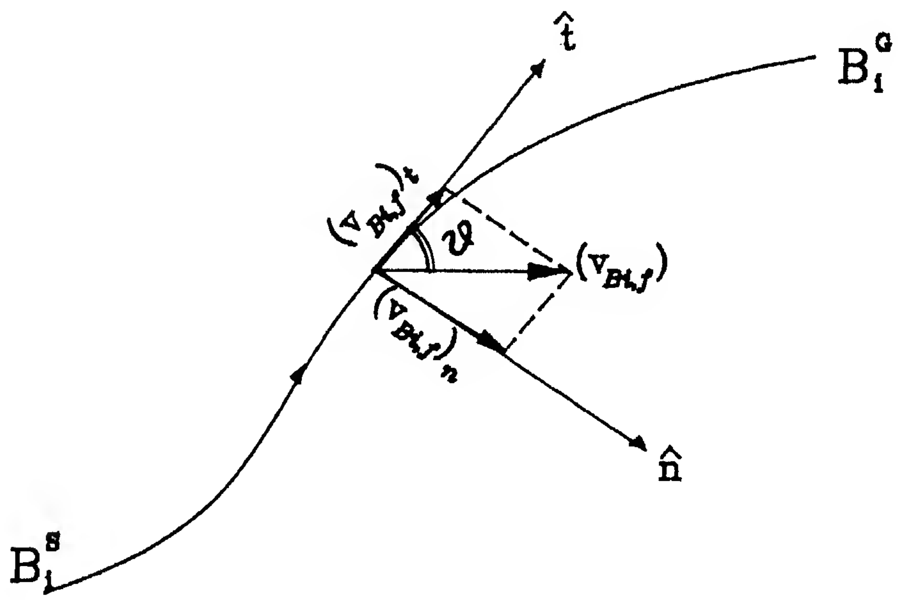


Figure 4.29: Tangential and Normal components on path

decided solely by $(v_{B_i})_t$, the tangential component of velocity. Let us assume that $\vec{v}_{B_i}$ subtends an angle $\vartheta$ with the $\hat{t}$ axis as shown in Figure 4.29. Then the velocity of escape for body $B_i$ can be rewritten as

$$\vec{v}_{B_i} = (v_{B_i})_t \, \hat{t} + (v_{B_i})_n \, \hat{n} \tag{4.13}$$

where the magnitudes of the tangential and normal vector components are given by

$$\begin{aligned} (v_{B_i})_t &= \|\vec{v}_{B_i}\| \cos(\vartheta) \\ (v_{B_i})_n &= \|\vec{v}_{B_i}\| \sin(\vartheta) \end{aligned} \tag{4.14}$$

For path-preference bodies, the velocity is decided on the basis of $(v_{B_i})_t \, \hat{t}$. If $(v_{B_i})_t \, \hat{t}$ goes beyond the limiting value $v_{\max}$, then the resulting velocity is set equal to the limiting value,

$$\|\vec{v}_{B_i,f}\| = v_{\max}.$$

Otherwise $\vec{v}_{B_i,f}$ is set equal to the tangential component of the escape velocity,

$$\vec{v}_{B_i,f} = (v_{B_i})_t \, \hat{t}.$$

Point-preference bodies do not have any path restriction. They are allowed to faithfully follow the escape velocity vector when faced with obstacles around themselves provided they do not violate the kinematic constraints.

## 4.6 Examples with Motion Constraints

The algorithm described in the preceding sections has been tested on an IBM-AT machine running at 10 MHz. The program is written in C and a few simulated examples are presented in this section to demonstrate the salient features of the algorithm.

### 4.6.1 Example with multiple goal points

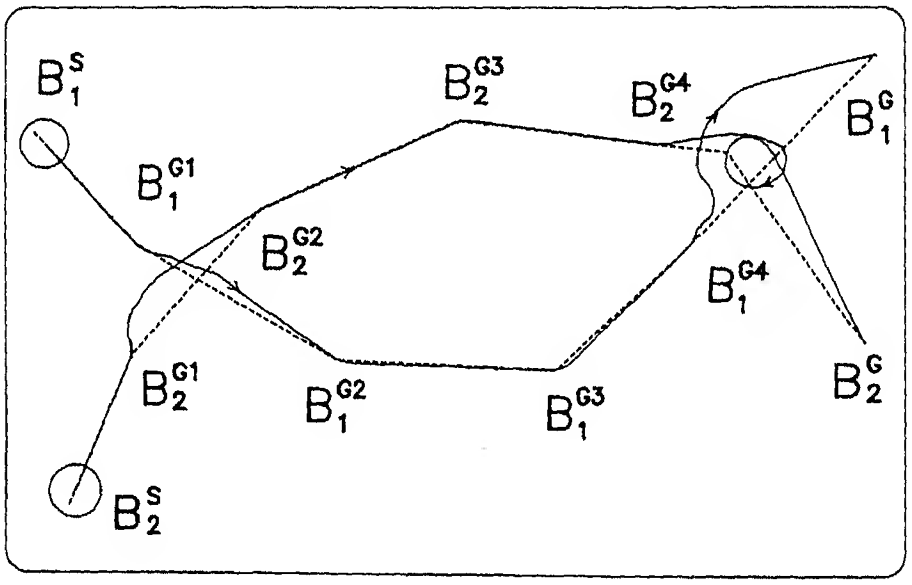


Figure 4.30: Demonstrating a point-preference problem

Figure 4.30 shows two bodies $B_1$ and $B_2$ situated near $B_1^S$ and $B_2^S$ respectively. The goal points are shown by the connected chains $B_1^{G1}$, $B_1^{G2}$, $B_1^{G3}$, $B_1^{G4}$, $B_1^G$ for body

$B_1$ and $B_2^{G1}$, $B_2^{G2}$, $B_2^{G3}$, $B_2^{G4}$, $B_2^{G}$ for body $B_2$. Both bodies are labelled as *point-preference* bodies. The continuous line is the path traced by the bodies during the run. The rule set used for this problem is shown in Table 4.8. Figures 4.31 and

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Distance between* **S** *and* **P** *is* | *Rel. Velocity Direction of* **S** *w.r.t.* **P** *is* | *Escape Velocity Direction of* **P** *w.r.t.* $\vec{v}_P$ *is* |
| *1* | Very Near | North-East | South-East |
| *2* | Very Near | South-East | South-West |
| *3* | Near | East | South |

Table 4.8: Set of rules used for example with two bodies

4.32 show intermediate stages of execution. It is observed that a collision between the bodies is avoided at the intersection of the goal points $B_1^{G1}$-$B_1^{G2}$ and $B_2^{G1}$-$B_2^{G2}$. A collision is again expected between goal points $B_1^{G4} - B_1^{G}$ and $B_2^{G4} - B_2^{G}$. It is seen that the bodies are able to successfully maneuver a turn here. A loop in the path of $B_2$ has occurred because in the first attempt to touch goal point $B_2^{G4}$, the body $B_1$ had come dangerously near the point. So body $B_2$ deviated from the path of $B_1$ and missed the goal point $B_2^{G4}$ in the first try. However in the second try, body $B_2$ is able to access the point $B_2^{G4}$ since by this time body $B_1$ had gone away from the zone of influence. The radius of the loop is decided by the maximum allowed turning radius,and the clockwise direction is a consequence of the framing of fuzzy rules. Figures 4.31 through 4.34 show different frames at different times of execution.

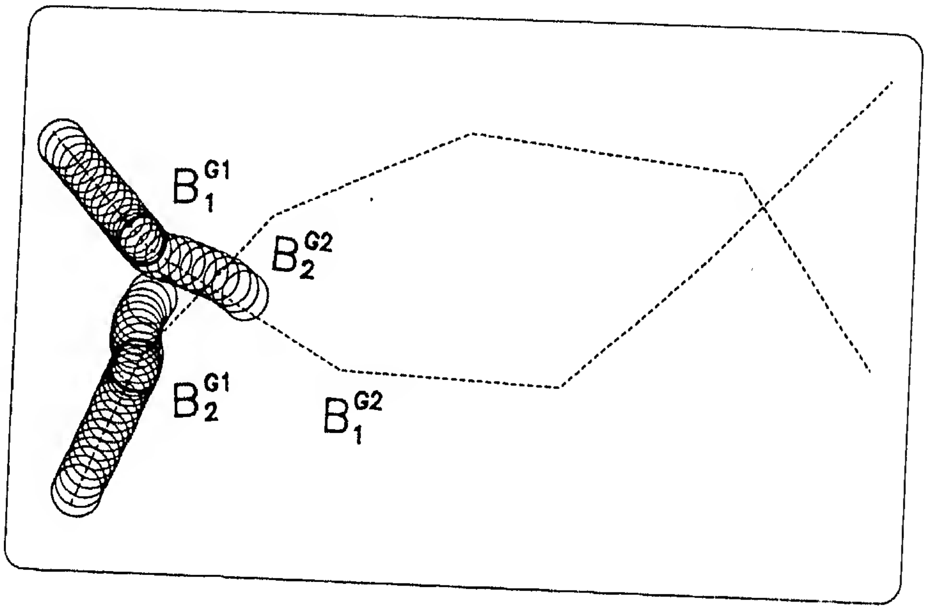


Figure 4.31: Point-preference problem : Frame 1

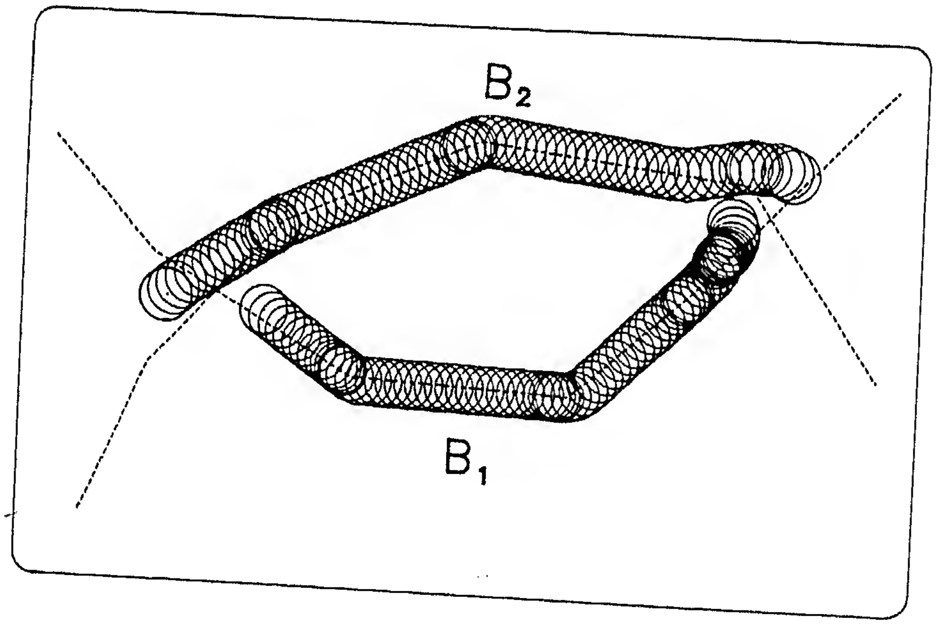


Figure 4.32: Point-preference problem : Frame 2

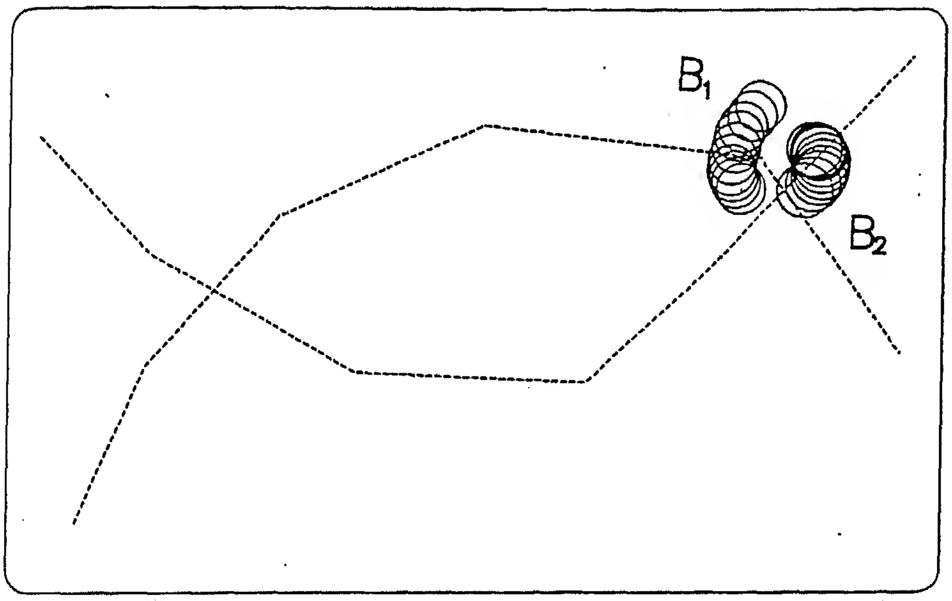


Figure 4.33: Point-preference problem : Frame 3

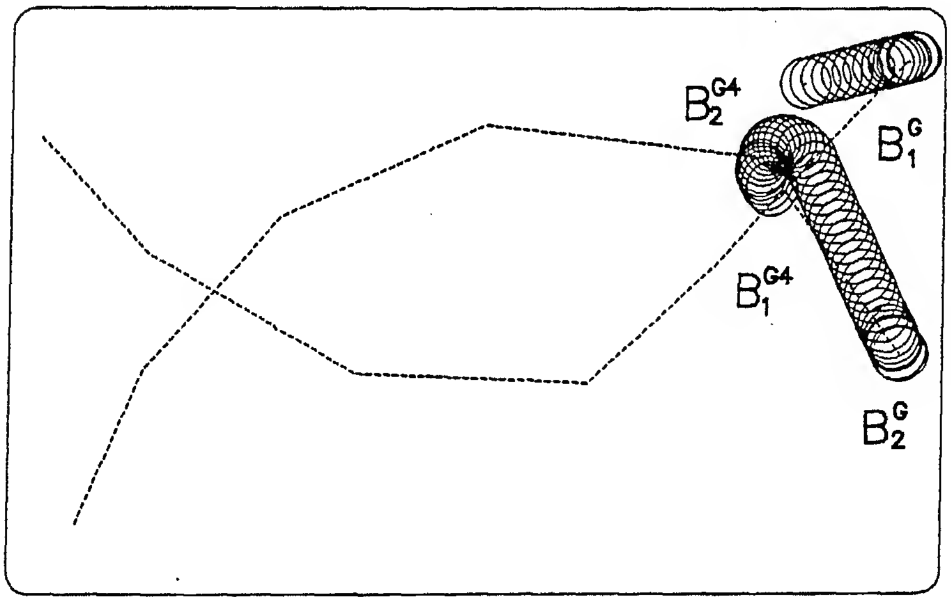


Figure 4.34: Point-preference problem : Frame 4

### 4.6.2 Examples showing various path-priorities

Figure 4.35 shows three bodies $B_1$, $B_2$ and $B_3$ which have started from their respective starting positions and are proceeding towards their respective goals. All bodies are point-preference bodies. The paths of the bodies are chosen such

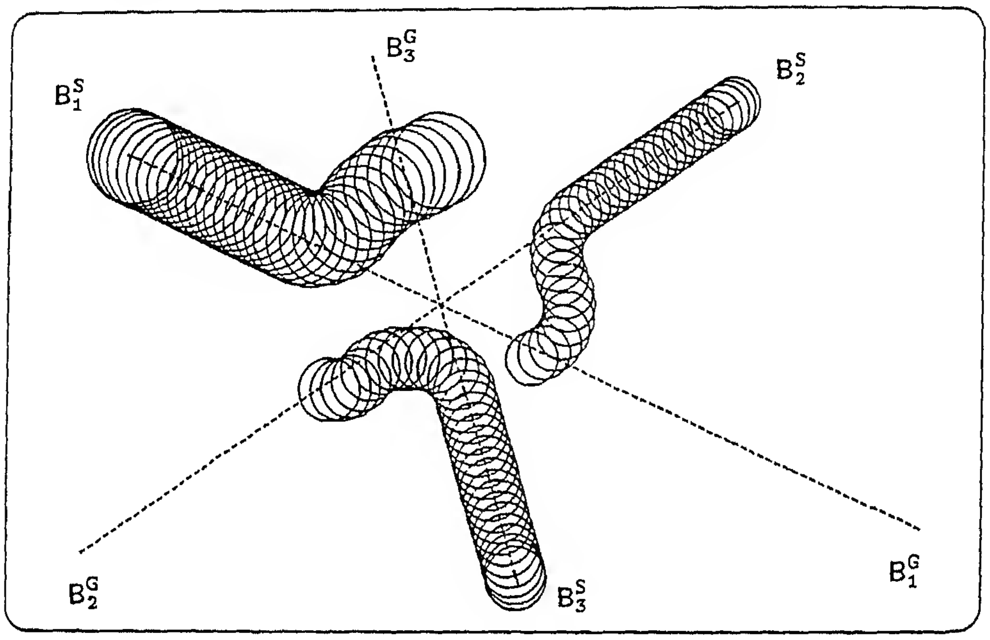


Figure 4.35: Point preference with common collision point : Sequences for first half

that they intersect at the same point. Figure 4.35 shows that the bodies start moving out as soon as they reach near the point of collision. This is where the bodies sense an obstacle "Near" themselves. The path taken by the bodies during the latter half of the execution is given in Figure 4.36. It is observed that the bodies proceed towards their respective goals along a straight line.

The entire path traced by the centre of the bodies is given in Figure 4.37. It is observed from the previous figures that the bodies effectively rotate clockwise about the central position in order to avoid collision. This is a consequence of the framing of rules, which assigns the preferential direction of escape. The velocity components of the three bodies are plotted in Figure 4.38 where the chain dotted

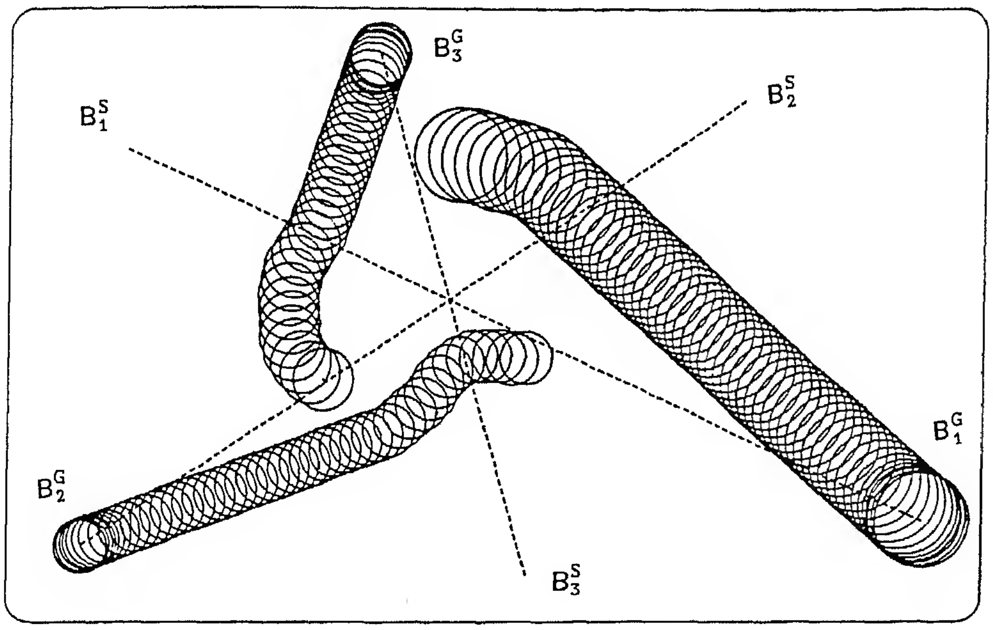


Figure 4.36: Point preference with common collision point : Sequence for latter half

line is the tangential component of velocity and the dotted line shows the normal component along the path. It is clear that the bodies take different times to complete the execution.

The earlier problem is slightly altered by specifying the body $B_2$ as a path-preference body. The collision free trajectories traced by the centers of the bodies is shown in Figure 4.39. The velocity plots are given in Figure 4.40. It is observed that body $B_2$ slows down for some time in order to allow the other two bodies to maneuver a turn. It is also seen that the deviation of body $B_3$ from the intended path is higher than depicted in Figure 4.37.

In the third example only body $B_3$ is a point-preference body. The other two bodies are path-preference bodies. The resulting motion of the bodies while they are maneuvering a turn is depicted in Figure 4.41. The trace of the centers of the three bodies is given in Figure 4.42. The velocity plots are given in Figure 4.43.

It is interesting to note that the body $B_3$, the only one with point-preference,

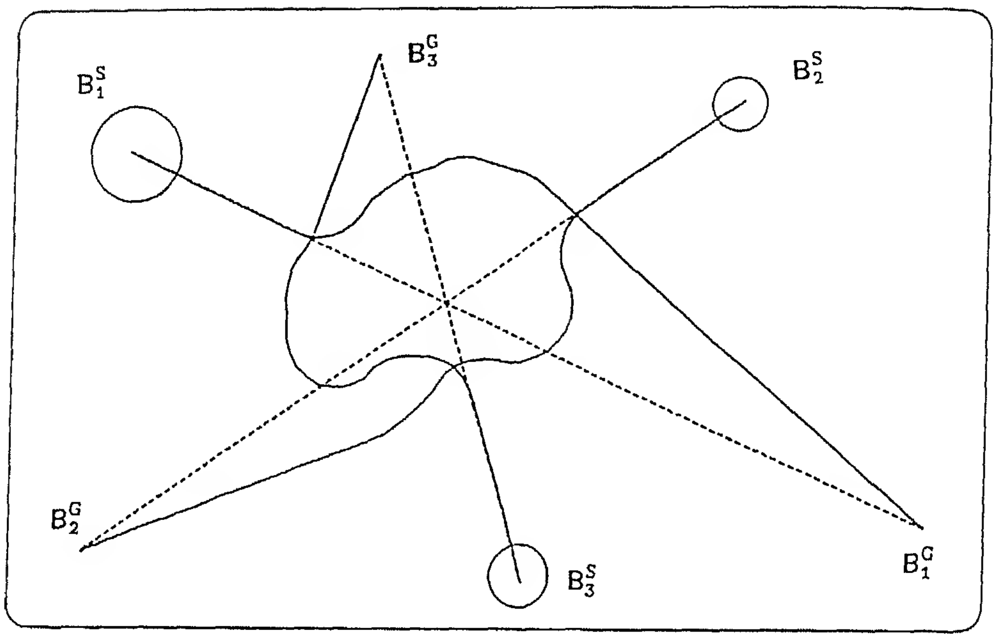


Figure 4.37: Point preference with common collision point : Path traced by the centre of the bodies

has to hunt somewhat to avoid a collision before it finds its way to the goal. The other two bodies have to slow down considerably to allow body $B_3$ to discover a safe course. It can be observed from Figure 4.43 that the time taken to complete the task is much more than the earlier cases.

The final example shows an application in which a body is to maneuver a turn through a set of circular static obstacles. The obstacles are represented through bodies $B_2$, $B_3$, $B_4$, $B_5$, $B_6$ and $B_7$ (see Figure 4.44). The body $B_1$ is a path preference body and has to move from $B_1^S$ to $B_1^G$. Since this problem does not involve too many moving objects, the fuzzy rules of Table 4.3 are applied here.

The motion of the body $B_1$ is decided by the following strategy. The primary intention is to find out a pathway between the bodies. This is achieved by deciphering the appropriate information from the fuzzy output vector. A typical output membership grade curve will look like the one shown in Figure 4.45. in which the segment BCD shows a "depression" in the curve. This depression is interpreted

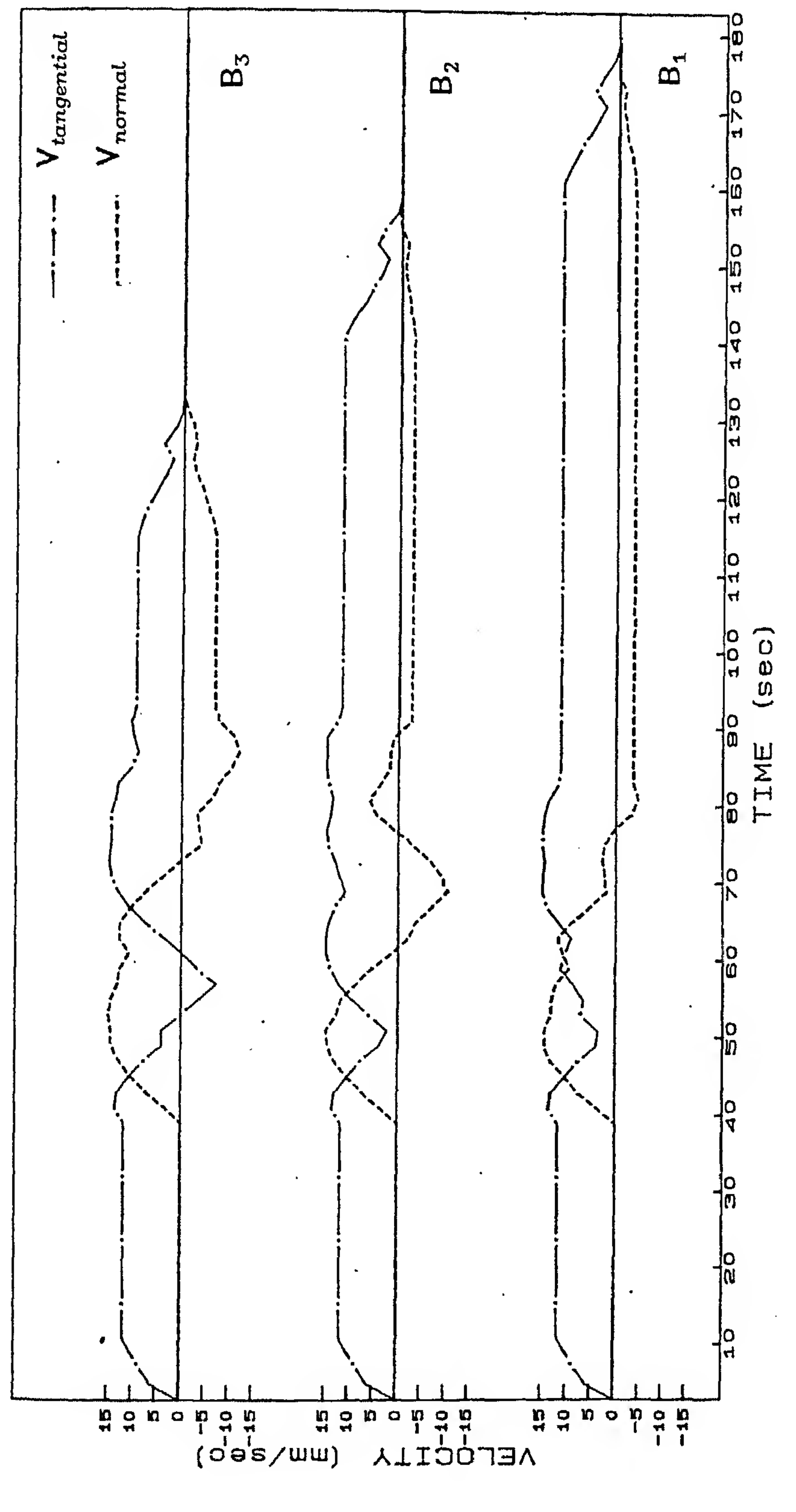


Figure 4.38: Point preference with common collision point : Plot of Tangential and Normal components of velocity

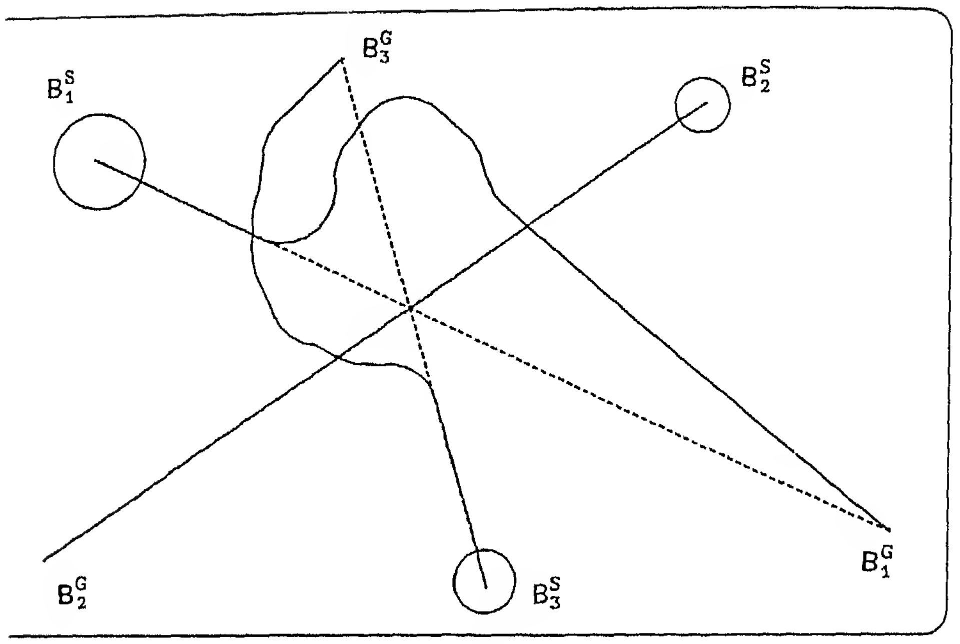


Figure 4.39: Trace of the centers for two point-preference bodies

s an open doorway which might be free for access. Thus the fuzzy output vector is scanned for such depressions and a motion is imparted in the direction which shows such a depression. When two depressions are discovered, the one which is deeper is chosen if it does not lead the object away from the goal. Otherwise, the depression which takes it nearer to the goal is chosen.

Figure 4.44 shows the path traced by the center of body $B_1$ when surrounded by seven bodies. It is obvious that this strategy assumes that the doorway is wide enough to allow the obstacle to enter through it. The algorithm just identifies that doorway based on sensor readings. This is however not a fool-proof method to proceed in practice because it will definitely fail if the doorway is smaller than the object size. It is hoped that, in practice, a physical bump will be signaled by tactile sensors and freeze all motion preventing any further attempt to pursue the same path. A method to find out an imminent collision is however possible through fuzzy logic. A very large value of the defuzzified output (i.e. a very high value of the c.g. of the output curve) should be interpreted as a sure sign of

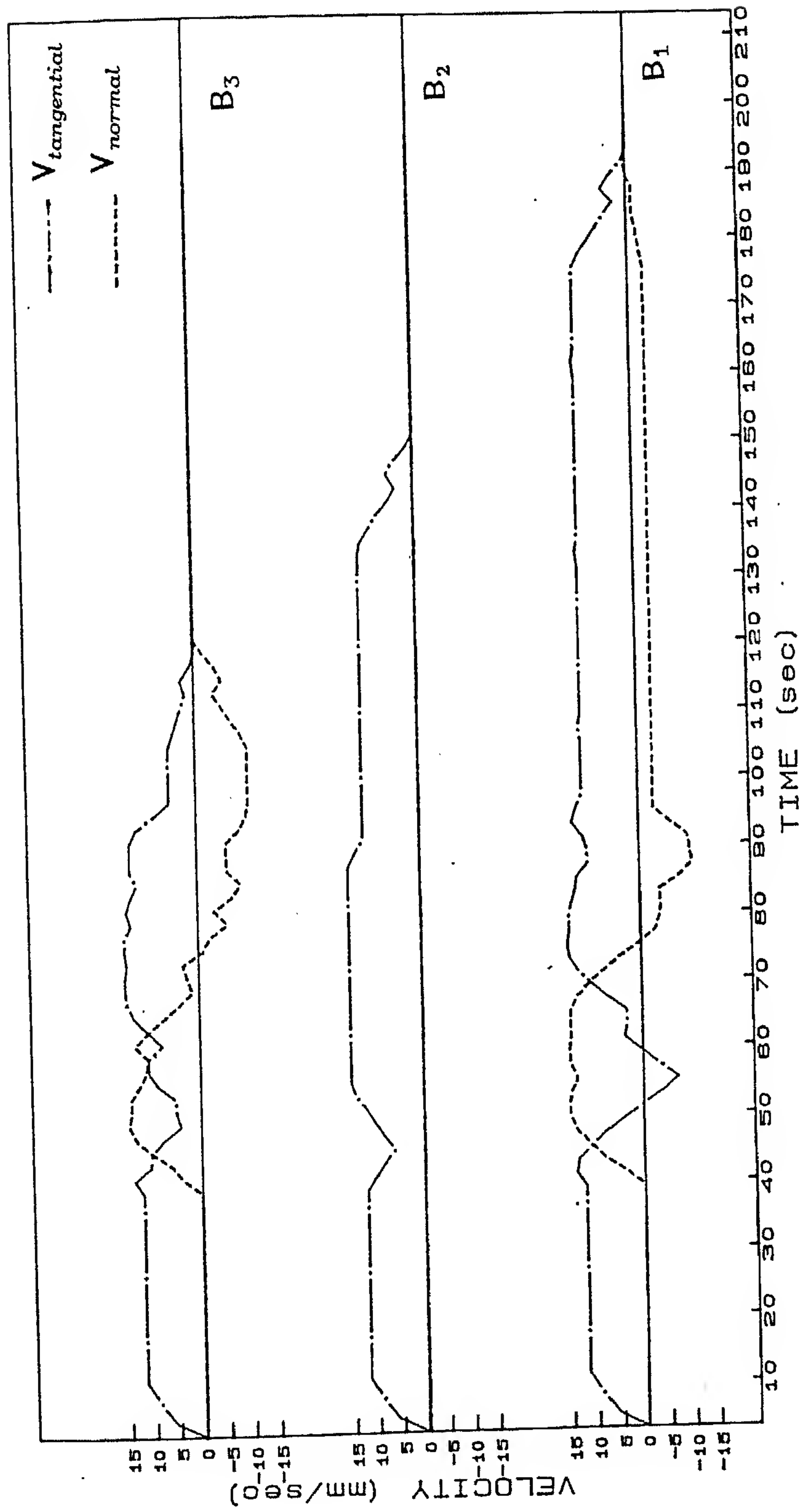


Figure 4.40: Velocity plots for two point-preference bodies

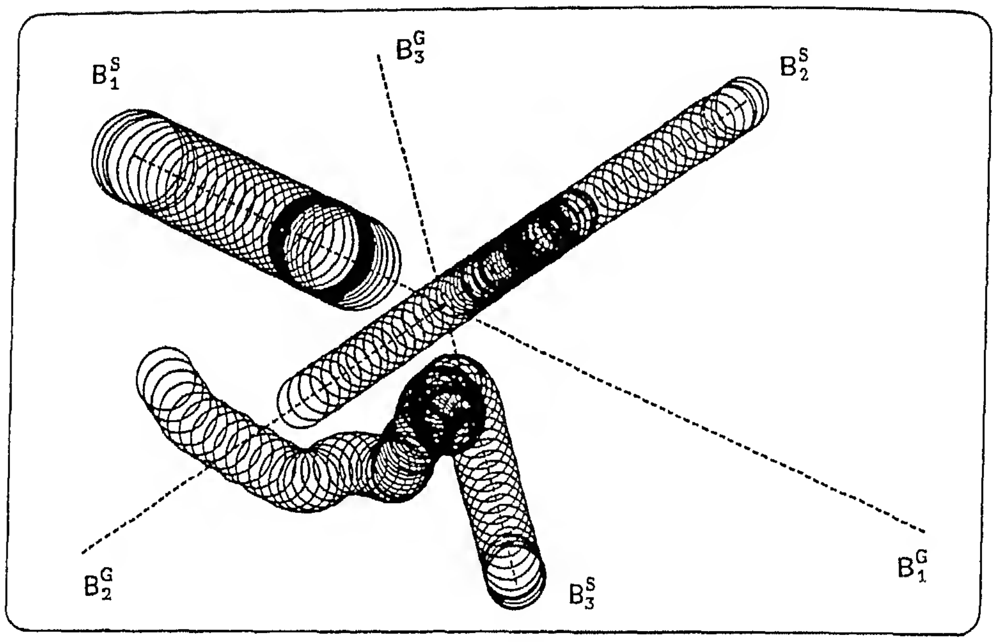


Figure 4.41: Motion of three bodies, with only one point-preference body

danger. When high values of the output are obtained, pursuance of the current goal is completely overpowered by the escape vector. (Note that the resultant of the two vectors decide the net motion of the obstacle.) Obstacle avoidance then becomes the primary aim compared to the goal task and the body automatically moves towards a direction that takes it away from the obstacles.

The execution time for the various problems solved in this chapter are presented now. The program for implementing this algorithm is written in C and works in the DOS environment on an IBM-AT working at 10MHz. The execution time in given in Table 4.9. It is observed from the table that the execution time increases when more number of rules are used in the fuzzy controller. This is evident from the first two rows of the table. Even though the calculations in the fuzzy domain increase two-fold with two rules, the overall increase is of the order of one-third. The variation of execution time on having more path-priority bodies may be assessed by inspecting rows 3, 4 and 5 of Table 4.9. It is observed that execution time increases marginally for cases having more path-priority bodies. This variation

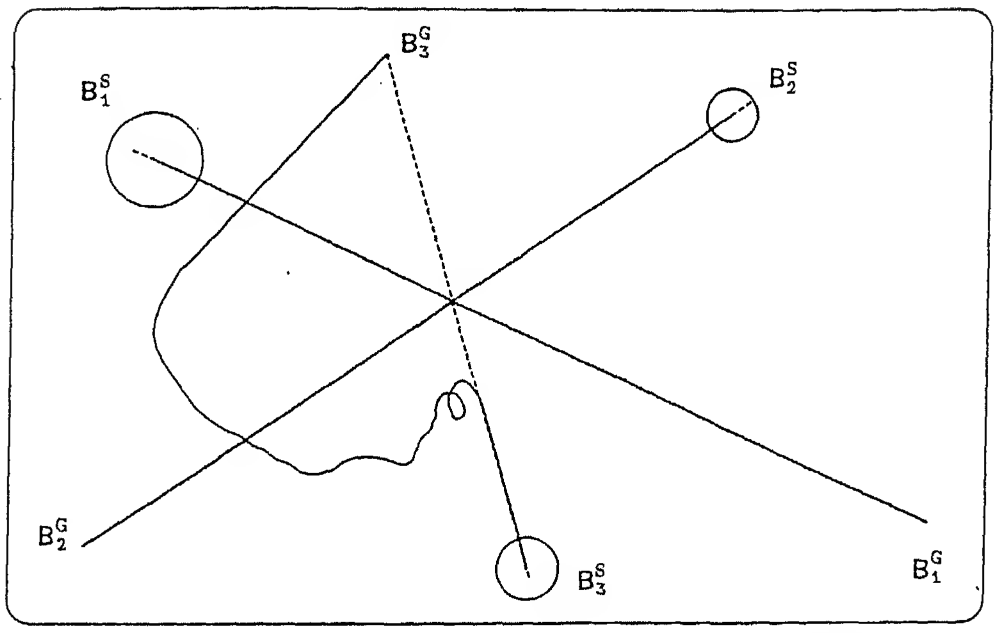


Figure 4.42: Trace of the centers : only one point-preference body

has occurred due to more calculations being done in the crisp domain for checking the components of the escape vector along the path, and trying to maintain the body on the specified path. Finally, the last column gives the execution time for the five-body problem solved in Section 4.4.5. It is observed that this problem requires the highest time to simulate because the algorithm's complexity increases as $n^2$ (where $n$ = number of bodies).

## 4.7 A Suggestion for Sensor Installation

It was remarked in Section 4.2 that the interaction between bodies and the assessment of their relative position will depend a lot on the mode of sensor installation. Since this topic itself covers a wide span, this chapter is concluded by giving a simple suggestion to work with sensors. This section should only be taken as a supplementary note because the facts mentioned here are subject to change depending on the type of sensors used. The other facts mentioned in the preceding

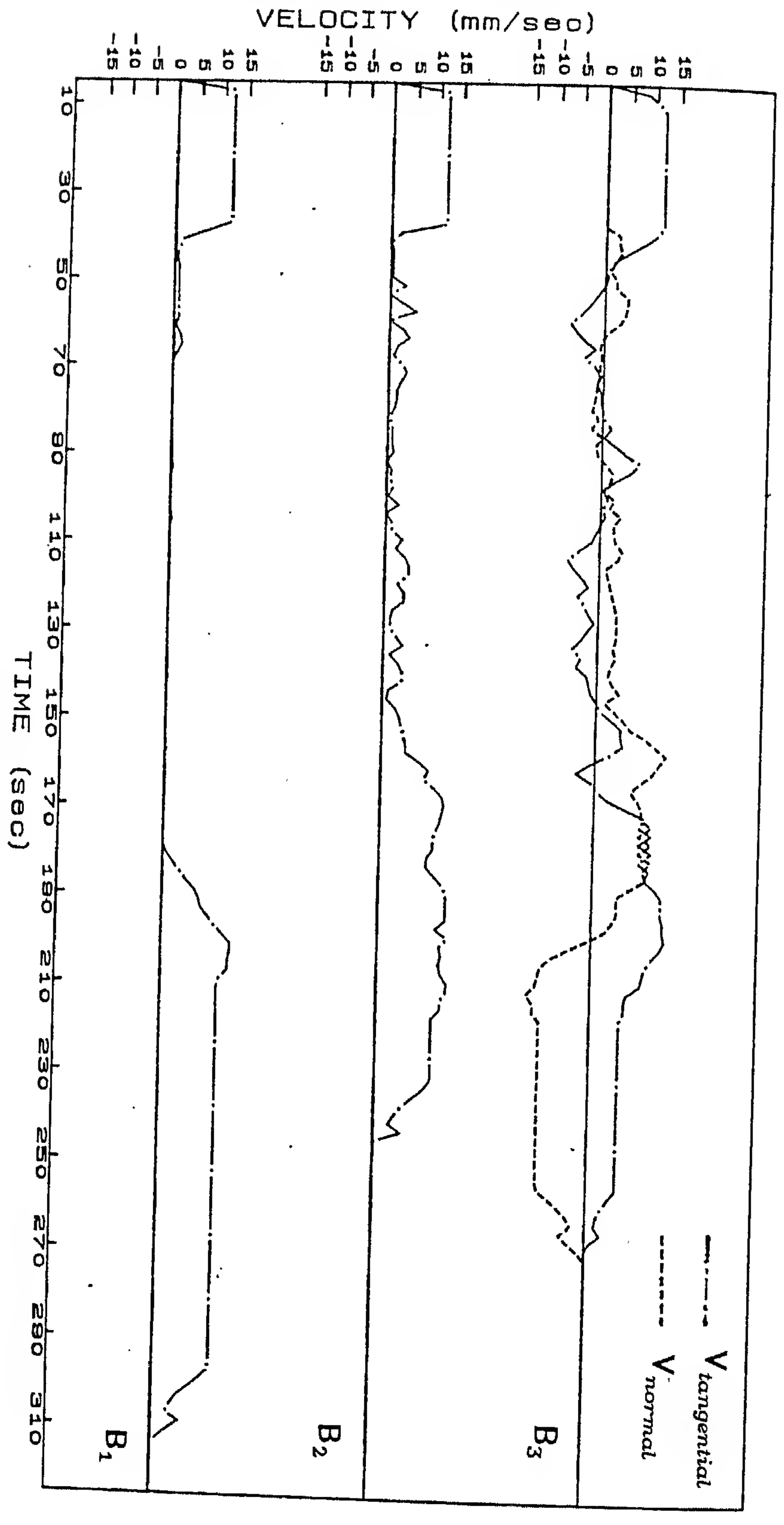


Figure 4.43: Velocity plots with only one point-preference body

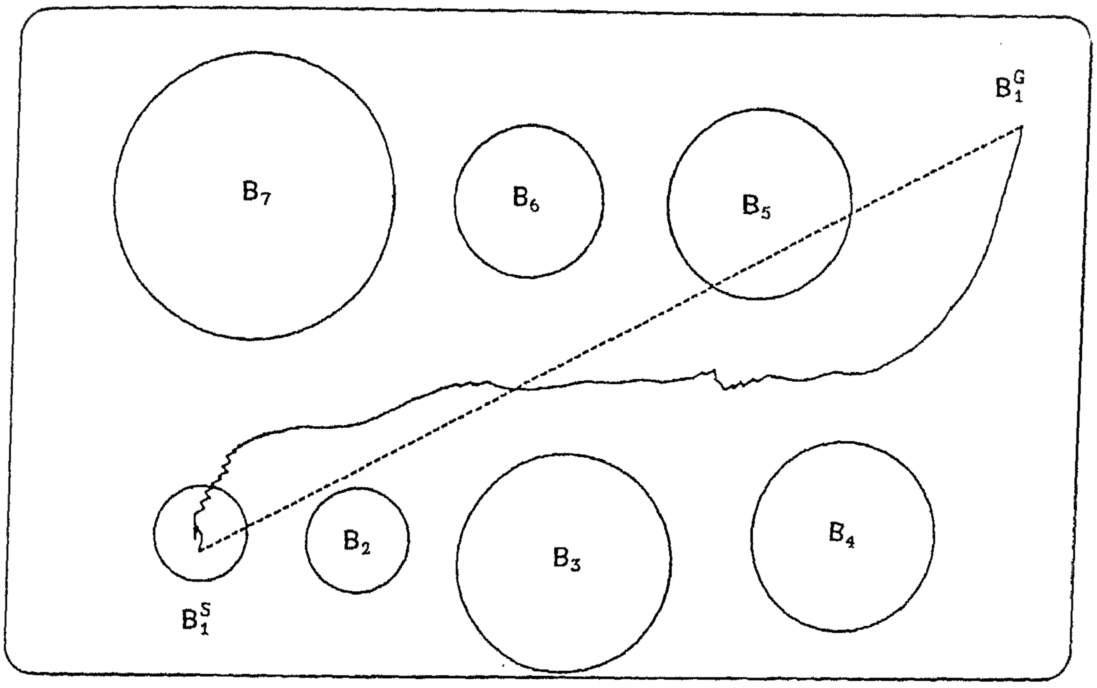


Figure 4.44: Finding a path through a set of static obstacles

sections will however still hold for the subsequent analysis.

This section assumes that ultrasonic ranging sensors are used around the body to detect obstacles. This is not an impractical choice considering that most experimental autonomous vehicles use ultrasonic sensors for distance measurement. The fact that ultrasonic sensors are not very accurate [28] gives credence to the use of fuzzy logic for data analysis. It may be noted that fuzzy logic is ideally suited to analyze inaccurate information through a robust fuzzy controller. So the proposal to use a fuzzy obstacle avoidance algorithm coupled with ultrasonic sensors seems to hold good promise.

Ultrasonic sensors have a *beam angle* within which they can detect an obstacle. The space which lies within the beam angle is called the *insonified region*. The number of sensors to be installed around the vehicle will depend on the beam-angle of each sensor (which is assumed to be same for all). For our simulation, it is assumed that each sensor has a beam angle of $\pm 30°$. Thus it is deduced that it might be sufficient to install 8 ultrasonic sensors around the periphery of each

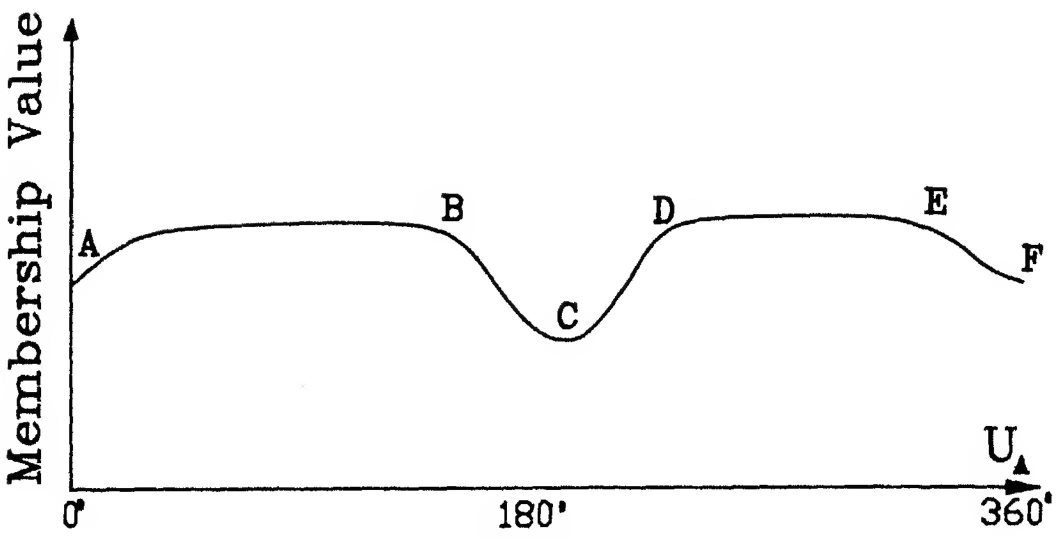


Figure 4.45: A membership curve showing a "depression"

vehicle to cover the entire range of 360 degrees (Actually only six sensors should be theoretically sufficient, but to allow safety eight sensors have been used. This ensures beam overlap for higher reliability.) The location of the ultrasonic sensors might be chosen to be the points 1, 2, 3, 4, 5, 6, 7 and 8 in Figure 4.46.

To keep track of the orientation of the vehicle with respect to the absolute frame $X_0$-$Y_0$, a *compass* is fixed on the vehicle. Although, traditionally all compasses point towards **North**, we choose to allow our compass needle to always point **East** ! This is done only for representational convenience because conventionally East represents 0 degrees. The absolute value of the compass reading then may be taken to the absolute orientation with respect to the base frame $X_0$-$Y_0$. Thus according to our convention, the compass needle will always be aligned along the $X_0$ axis of the frame coordinate system.

Let us suppose that one of the sensors detects an obstacle. Considering Figure 4.46 the sensor located at position '3' detects an obstacle. It is now required to estimate the position of the obstacle with respect to the body in the absolute frame. Let us assume that $\phi$ is the angle subtended by the front vector $\hat{f}$ of the vehicle with respect to *East* of the compass, i.e. w.r.t the absolute frame. Since the position of sensor '3' is always fixed with respect to the front vector $\hat{f}$, of the

| *Rules* | *Bodies* | *Execution time per step (sec)* |
|---|---|---|
| 1 | 2 | 0.311 |
| 2 | 2 | 0.422 |
| 3 | 3 | 1.932 (no path-pref. bodies) |
| 3 | 3 | 2.083 (one path-pref. body) |
| 3 | 3 | 2.331 (two path-pref. bodies) |
| 3 | 5 | 3.333 |

Table 4.9: Execution time for various combination of rules and number of bodies

vehicle, let $\psi$ represent the angle subtended by the activated sensor with respect to the front. Then the angle subtended by the obstacle in the absolute frame, $\theta$, is given by

$$\theta = \phi + \psi$$

The value of $\theta \approx 135°$ represents *North-West* in Figures 4.4 and 4.46.

In an unknown environment, without any knowledge about the presence or motion of the neighbouring bodies, it might be difficult to identify a secondary body. This is a practical problem which has been ignored in this thesis. Details of estimating positioning and world mapping are discussed, for example, in [2,25,28]. Herein, we suppose that the vehicle's controller assumes that there are at most eight secondary bodies around the vehicle, one situated at each sensor's field of view. When a sensor triggers, it is assumed that the secondary body is located along the normal to the sensor as shown in Figure 4.47. Though this is a simplified representation, it is practical considering that ultrasonic readings are quite inaccurate. The other alternative is to combine adjacent sensory readings to give the estimated position of one common obstacle, but this method will sometimes lead to larger anomalies in positioning when the sensor normals are not parallel and the separation between the sensors is large. If sensors are placed closely (as in [28]), then it is sensible to combine adjacent sensor readings for assessing the exact position of the surrounding obstacles.

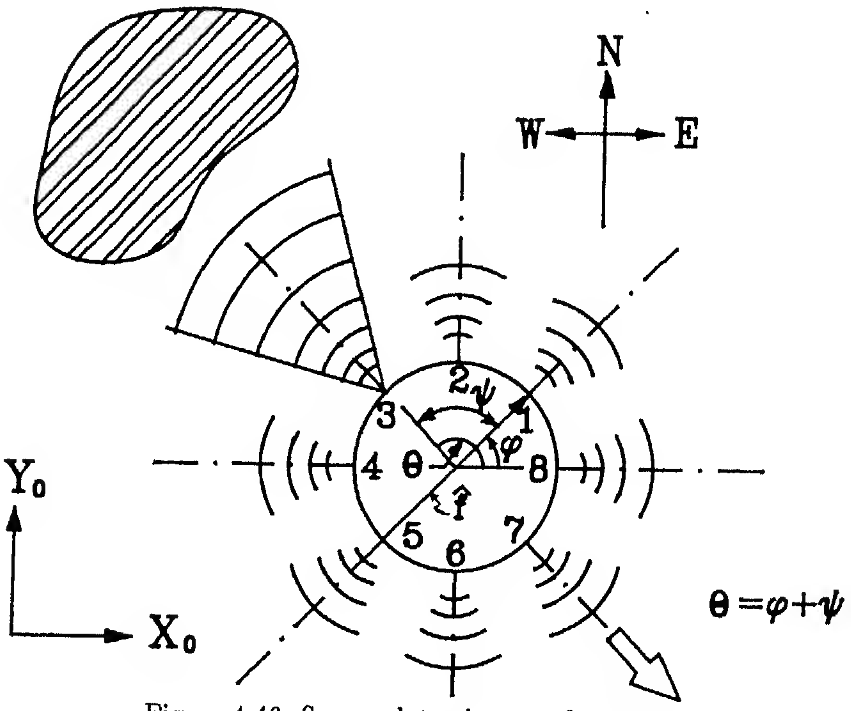


Figure 4.46: Sensor detecting an obstacle

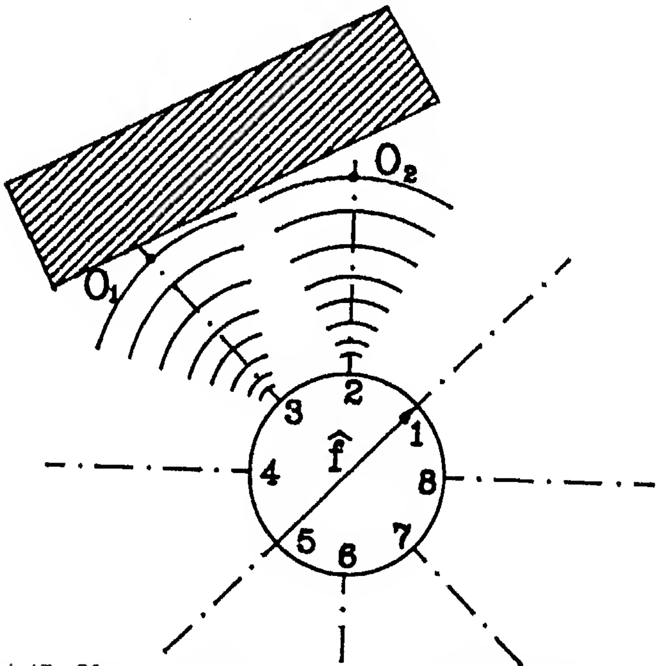


Figure 4.47: Obstacle being assumed to be along the sensor normal

*This very remarkable man(*ipulator*)*
*Commends a most practical plan;*
*You can do what you want*
*If you don't think you can't,*
*So don't think you can't if you can.*
– Rev. Charles Inge

Chapter 5

# Motion Planning of a Planar Manipulator in a Dynamic Environment

## 5.1 Introduction

A common application of collision avoidance algorithms is the motion planning of a robot manipulator amongst obstacles. A manipulator is essentially a *chain* of bodies satisfying a definite kinematic relationship. It is thus natural to extend the algorithm developed in Chapter 4 for individual bodies, to path planning of a chain of bodies amongst obstacles. In the earlier problem each body was assigned a separate objective, whereas in the current problem the chain of bodies have a common objective — that of satisfying a particular end-effector trajectory while avoiding obstacles. This problem is again solved assuming that no prior information about the environment is available to the controller of the manipulator, and the collision avoidance strategy is based solely on the assessment of nearby obstacles through sensors mounted on the surface of the links. Hence, the existence, path and the velocity of approach of an obstacle (if any) in the immediate vicinity of the manipulator, is deduced on-line.

In this chapter various ways of solving the manipulator configuration are discussed. These are the algebraic methods when there are no obstacles around, the geometric methods when the algebraic solution fails near singularities, the geometric methods of reorienting links keeping the end-effector static when the link is influenced by the obstacle, and purely geometric ways of moving the entire manipulator away from an obstacle when a collision is imminent. The objective of collision avoidance while maintaining the end-effector on the desired trajectory can be achieved only when the manipulator is given some liberty to adjust its links without influencing the end-effector's position. This requires a certain amount of *redundancy* in the manipulator's degrees of freedom. We thus consider a manipulator having extra degrees of motion to provide for the added flexibility needed for collision avoidance. In this chapter only planar manipulators having rotary joints are analyzed.

Since the behaviour of the obstacle(s) in the environment is completely unknown, planning a global path with collision avoidance before execution begins is not possible. In the absence of prior knowledge, the end-effector is initially directed to move towards the goal along a straight line which is the shortest possible path. Even though the initial default path is assigned this way, the manipulator configuration and the end-effector location may change later depending on the presence of obstacles. At times the obstacles can come dangerously near the manipulator resulting in an imminent collision. If the sensors on the manipulator indicate such a possibility, the manipulator is made to sacrifice its main objective of maintaining the end-effector on its trajectory and an attempt is made to satisfy the primary objective of collision avoidance. Under such circumstances, the links of the manipulator deflect to avoid the obstacle. The manipulator is made to resume its old trajectory after the obstacle has passed. If the obstacle happens to stay in the vicinity of the manipulator, the arm is made to go through a sequence of positions to circumvent the obstacle, and bring the end-effector to its old trajectory by approaching from a different direction. The decision to halt the manipulator under dangerous situations and to circumvent obstacles is taken automatically by the program. This chapter considers an example and examines the sequence of steps necessary to detour the obstacle.

## 5.2 Problem Statement

The objective of this chapter is to develop a strategy for moving the end-effector from a known starting position $E^S$ to another known goal position $E^G$. Based on the restrictions put on the end-effector's position on the path, this problem can be categorized into the following two groups:

1. The **point-to-point programming** problem
2. The **path-preference** problem.

In the point-to-point programming problem the path from $E^S$ to $E^G$ is not important and the end-effector is made to trace the shortest collision free path between the two points $E^S$ and $E^G$ by default. In the path-preference problem the end-effector is required to trace a specified path from $E^S$ to $E^G$. The second category is labeled as a *path-preference* problem because at times it may become absolutely necessary to deviate from the prescribed path to satisfy the paramount objective of collision avoidance. However, after adjustments of the manipulator configuration to avoid the collision are complete, the end-effector goes back to the point from where it had departed from the assigned path, perhaps from a different direction. Our use of the word "path-preference" gives the end-effector the liberty to deviate from the intended path under extremely dangerous situations.

Figure 5.1 shows a redundant manipulator with $n$ links connected by revolute joints. The $i^{th}$ link is denoted by $L_i$, its length by $a_i$ and width by $b_i$. The point at which link $L_i$ is connected to link $L_{i-1}$ (i.e. the location of joint $i$) is called the head of link $i$ and is denoted by $h_i$. Similarly the point of connection between $L_i$ and $L_{i+1}$ is called the tail of $L_i$ and is denoted by $t_i$. Thus, $h_i = t_{i-1}(i = 1, n)$. The base of the manipulator, B is, therefore, defined by $B = h_1$. The end-effector position is denoted by $E = t_n$, which is defined as the tail of the last link. Each joint is considered to have limited range of movement constrained by $(\theta_i)_{lower} \leq$

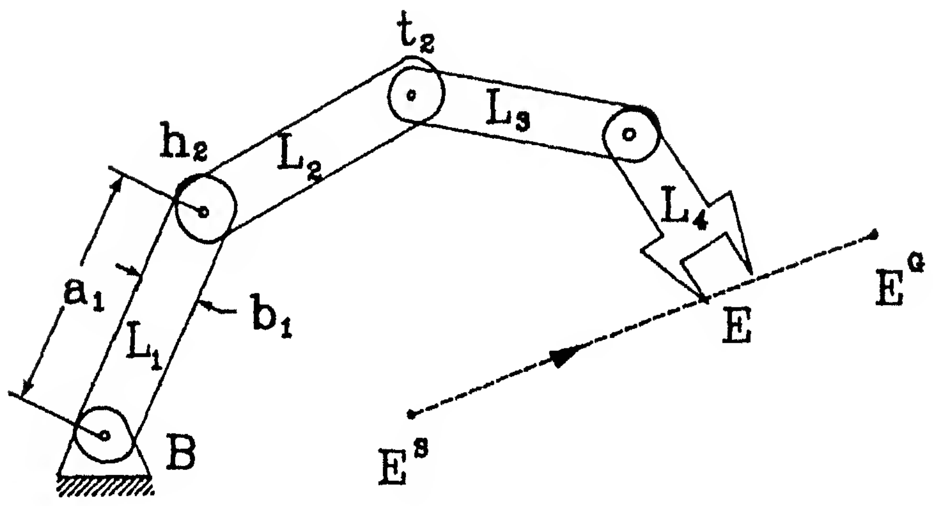


Figure 5.1: Description of a manipulator

$\theta_i \leq (\theta_i)_{upper}$. The kinematic constraints on the joint velocity and acceleration also require that $|\dot{\theta}_i| \leq \omega_{i,max}$ and $|\ddot{\theta}_i| \leq \alpha_{i,max}$.

## 5.3 Fuzzy Rules for Collision Avoidance of Links

The application of fuzzy rules will be greatly dependent on the perception of the world. It is supposed that this perception will be obtained from readings taken by on-board sensors mounted on the surface of the links. The first step is thus to convert the raw sensory readings to an appropriate model of the world. Because of the inherent inaccuracies of the sensors, it is likely that the model might not be a true representation of the actual world. The common methodology is thus to model the world as primitives comprising of either points, lines and planes along with their associated uncertainties, and work on this perceived world model to detect collisions and plan a collision free path. In this work the perceived world model is assumed to be approximate and comprising of only points. The associated uncertainties are taken care of by fuzzifying the measurements over the perceived world model and converting distances and angles into symbolic notation. The output quantity for a collision avoidance problem is also obtained as a fuzzy quantity which is finally converted to a crisp value. This method is believed to be effective in dealing with the associated uncertainties of sensory data.

The key point that needs to be highlighted is that fuzzy logic will be applied to a perceived world model and not on the actual raw data. The perceived world model consists of a set of points on a plane. These points are considered as *point obstacles* in space represented as $O_1, O_2, \ldots, O_m$. This chapter discusses an algorithm for avoiding collision of manipulator links with the point obstacles $O_j$ ($j = 1, \ldots, m$). The generation of the point obstacles from raw sensory data is a separate issue which depends on the type of sensors used. A suggestion for building the perceived world model from ultrasonic sensors mounted on the side of links is discussed in Section 5.8. A scheme for mounting ultrasonic sensors on the side of links is also suggested in Section 5.7.

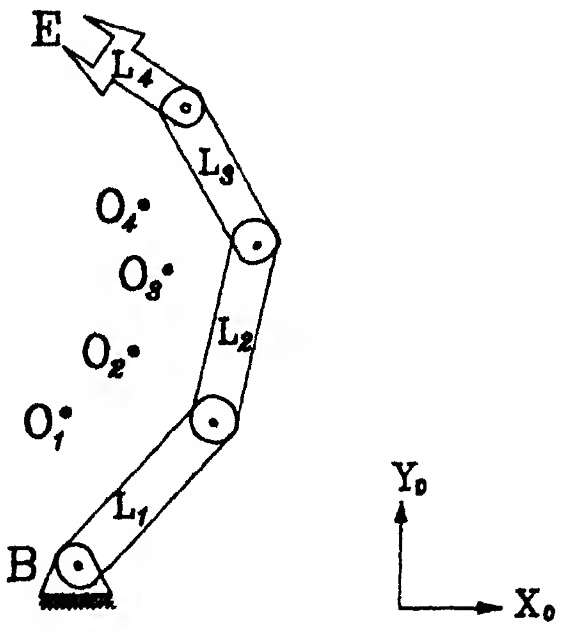


Figure 5.2: A manipulator amongst its point obstacles in perceived world model

Figure 5.2 shows a manipulator near the *point obstacles* $O_1$, $O_2$, $O_3$ and $O_4$. The coordinate system $X_0$-$Y_0$ is attached rigidly to the plane of motion. It is desired to find the influence of these obstacles upon the motion of manipulator. The obstacles are believed to have a repulsive action on the links of the manipulator.

The influence of the obstacles on the entire manipulator is now broken down into the individual effect of the obstacles on each link of the manipulator. Thus, each point obstacle $O_j$ is paired with each link $L_i$ of the manipulator to compute the influence on the link $L_i$. The net effect on the manipulator will be the resultant influence of all the obstacles on all the links of the manipulator[1].

Figure 5.3 shows a link $L_i$of the same manipulator. The head $(h_i)$ and tail

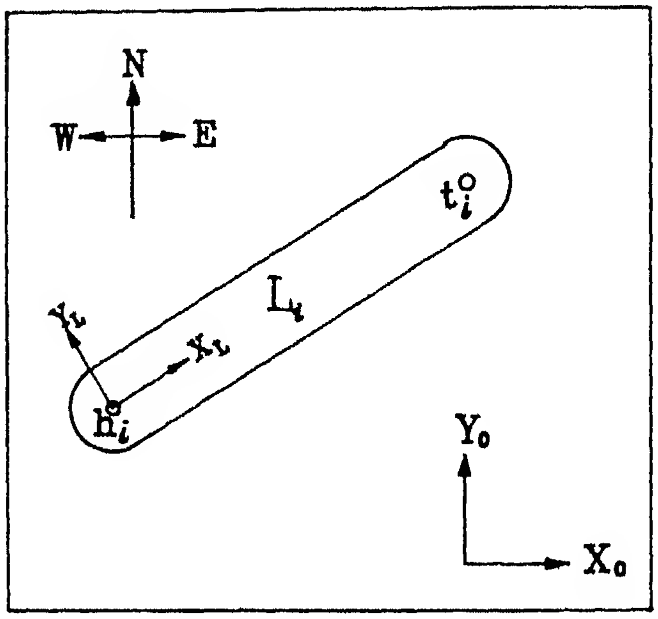


Figure 5.3: Link in original frame

$(t_i)$ of the link is also shown in the figure. A coordinate system $X_L$-$Y_L$ attached rigidly to the link at the head $h_i$. The coordinate system $X_0$-$Y_0$ is also shown in the figure. We now intend to introduce a rotation of frame analogous to the one introduced in Section 4.3.2. In this problem, the frame $X_0$-$Y_0$ is rotated so that it aligns itself with $X_L$-$Y_L$ — the link coordinate system. In the rotated frame $X_0^R$-$Y_0^R$ (see Figure 5.4), the point obstacle is unlikely to appear towards the *East*

[1]Although this strategy suggests an algorithm of order $\mathcal{O}(mn)$, it will be shown later in Section 5.8 that all obstacles need not be paired with all links. Obstacles will be shown to be *link specific*, i.e. each link will perceive only a few of the point obstacles as dangerous and ignore the rest.

or the *West* of the frame because (except the terminal links) the sides of the link will be occupied by its adjacent links.

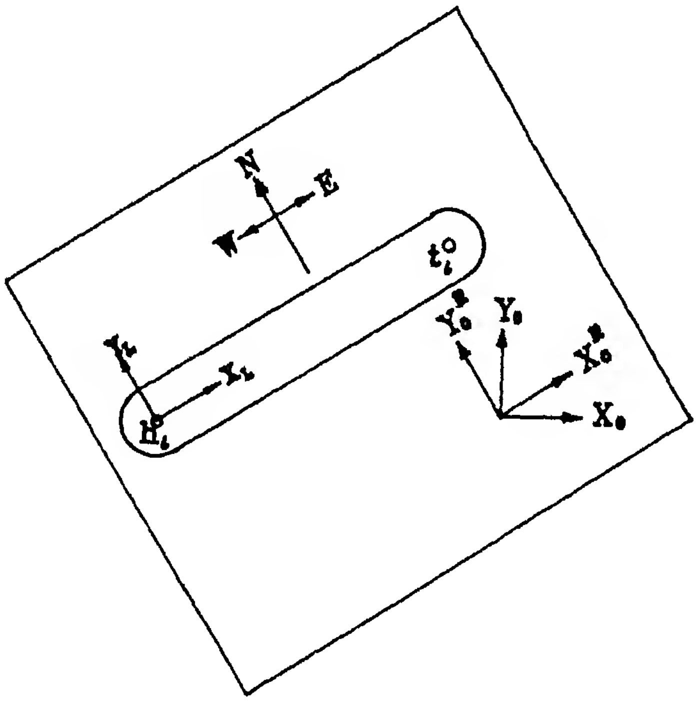


Figure 5.4: Link in rotated frame

To view the image obstacle from the link $L_i$, we set up two *observation points* on the link — at $P_{1,i}$, and $P_{2,i}$ as shown in Figure 5.5. Point $P_{1,i}$ is at a distance $s_1 a_i$ units from $h_i$ towards $t_i$ while $P_{2,i}$ is at a distance of $s_2 a_i$ units from $h_i$ in the same direction. The choice of the parameters $s_1$ and $s_2$ is presently assumed to be arbitrary, but later in Section 5.7.2, a formal way a deriving these values is discussed. For the present purpose let us assume a value of $s_1 = 0.2$ and $s_2 = 0.8$. From each observation point, the distance to the obstacle $O_j$ and its angle with respect to $X_L$ is calculated by geometric means. Thus in Figure 5.5, if $P_{1,i}$ is the current observation point then the distance $d_{P_1,ij}$ and the angle $\phi_{P_1,ij}$ is found from

$$d_{P_1,ij} = \sqrt{((P_{1,i})_x - (O_j)_x)^2 + ((P_{1,i})_y - (O_j)_y)^2} \qquad (5.1)$$

$$\begin{aligned}\phi_{P_1,ij} &= \text{atan2}((O_j)_y - (P_{1,i})_y \text{ , } (O_j)_x - (P_{1,i})_x) \\ &\quad -\text{atan2}((t_i)_y - (P_{1,i})_y \text{ , } (t_i)_x - (P_{1,i})_x)\end{aligned} \tag{5.2}$$

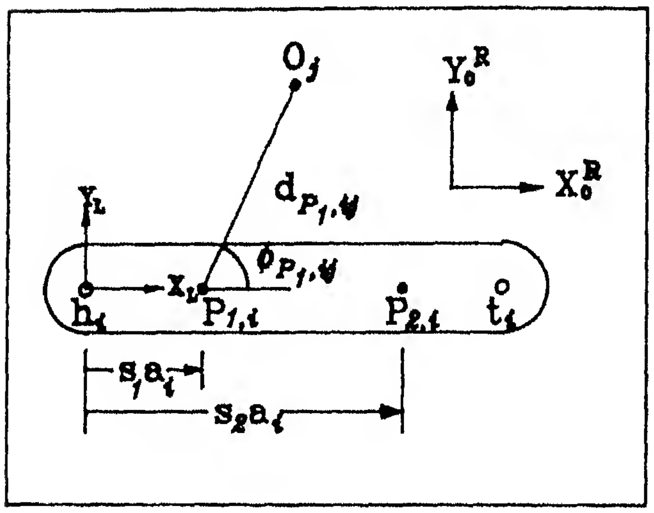


Figure 5.5: Measuring distance and angle from observation points

The quantities $d_{P_1,ij}$ and $\phi_{P_1,ij}$ are now fuzzified in accordance with the procedure outlined in Section 3.5. Table 5.1 lists the fuzzy rules applicable in the rotated frame $X_0^R$-$Y_0^R$, in tabular form. The rules are formulated so that the direction of escape of the links is exactly opposite to the direction of the point obstacle. The details of implementation and interpretation, are discussed in detail in Sections 3.5, 4.3.1, 4.3.2 and 4.4.3.

At the first sight the rules of Table 5.1 may not look to be an intelligent way of escape because it does not encode an effective way of *detouring* an obstacle. It is to be noted that the manipulator is a very constrained system because of its kinematic constraints. Since the links are more free to move *perpendicular* to the central axis of the links than *along* the line of axis (for rotary jointed manipulators and excluding the end-effector), it is more natural to frame the rules so that they avoid the obstacles by moving directly opposite to the direction of approach. This method will at least ensure that the link attempts to move away from the most likely point of collision. There might be more intelligent ways of escape for the end-

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Obstacle Distance is* | *Obstacle Angle is* | *Move Observation Post* |
| *1* | Near | North | South |
| *2* | Near | South | North |
| *3* | Near | North-East | South-West |
| *4* | Near | North-West | South-East |
| *5* | Near | South-East | North-West |
| *6* | Near | South-West | North-East |
| *7* | Quite Near | East | West |
| *8* | Quite Near | West | East |

Table 5.1: Set of rules for collision avoidance of links

effector (or for prismatic joints), for example, one which pushes the link towards the base B, of the manipulator to avoid an obstacle. This strategy will essentially retract the entire manipulator towards the base for detouring an obstacle. In Section 5.5.3 a new set of rules are given to achieve the above objective with an example.

The application of the fuzzy rules produces an escape motion vector for each observation point on a link. Let us represent the escape motion vectors (after defuzzification according to (2.30)) as $\vec{v}_{P_1,i}$ and $\vec{v}_{P_2,i}$ respectively for observation points at $P_{1,i}$ and $P_{2,i}$ respectively on link $L_i$. The cumulative effect of the escape motion vectors on the entire manipulator is shown in Figure 5.6. The escape motion vectors will be used to infer an appropriate motion sequence for the links in order to avoid collision.

## 5.4 Incremental Movement Schemes for a Manipulator

In accordance with the objective of the problem stated in Section 5.2, the manipulator is moved incrementally so that the end-effector traces its desired trajectory.

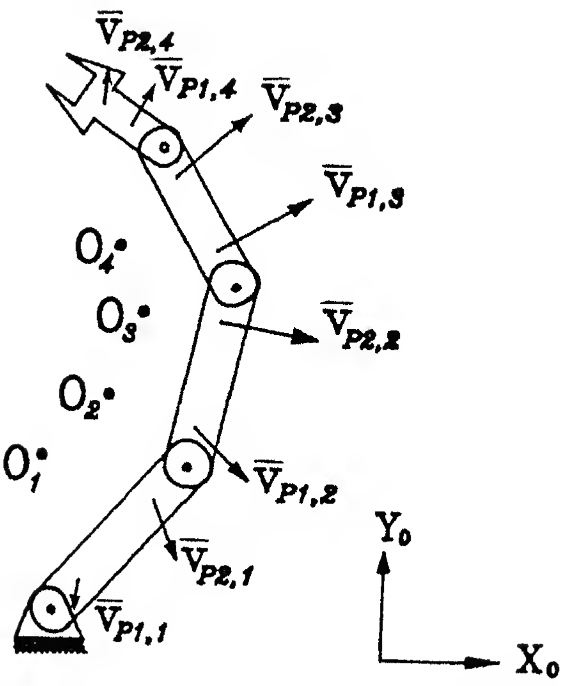


Figure 5.6: Escape motion vectors on each link of the manipulator

In path-preference problems the trajectory is provided by the operator, while in point-to-point problems the initial trajectory is the straight line that takes the end-effector to its subsequent goal point from its current position. When the manipulator is trapped near an obstacle, the links which lie close to the obstacle reorient themselves to avoid an impending collision. The reorientation is done so that the end-effector is kept static while the other links move away from the obstacle. This realignment of links is done by simple geometric manipulation. In this section we investigate various steps involved in moving the manipulator incrementally when the subsequent end-effector point is known. The problem is subdivided into two cases. In the first case the motion of the manipulator moving under the influence of external point obstacles is discussed. When the manipulator is in an obstacle-free environment, a suitable solution strategy for moving a redundant rotary manipulator must be used. This strategy is presented next for the sake of completeness.

### 5.4.1 Movement in the presence of obstacles

The derivation of the escape motion vector at the two observation points of each link was presented in Section 5.3. We now investigate methods to move the individual links using the escape motion vectors so that a collision can be avoided.

It will be convenient to develop a motion sequence if the escape motion vectors are acting only at the *joints* of the manipulator and not anywhere else. Thus the escape motion vectors acting at $P_{1,i}$ and $P_{2,i}$ are distributed proportionally to the ends $h_i$ and $t_i$. The effective motion vectors acting at the head and tail are equivalent to those acting at the observation points.

#### Apportioning of escape motion vector towards the joints

Consider Figure 5.7 where the escape vector $\vec{v}_{P_{k},i}$ $(k = 1, 2)$ is shown acting at point $P_{k,i}$ which is at a distance of $s_k a_i$ from $h_i$. ( The values $s_1$ and $s_2$ are now being referred to as $s_k (k = 1, 2)$. ) This vector is apportioned to the ends of the link,

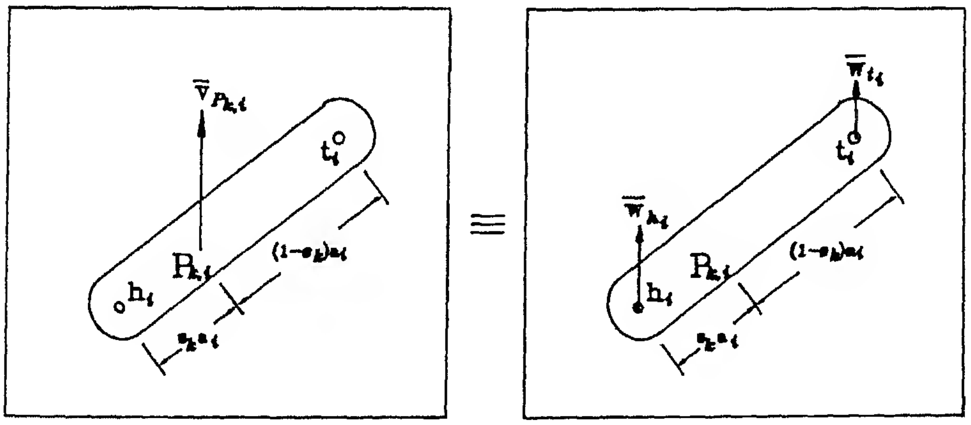


Figure 5.7: Apportioning of escape motion vectors towards the link ends

one acting at the head $h_i$ and another acting at the tail $t_i$. The component acting at the head is labelled $\vec{w}_{h_i}$ while that acting at the tail as $\vec{w}_{t_i}$. The magnitude of the components are given by

$$\|\vec{w}_{h_i}\| = \varrho(1 - s_k)\|\vec{v}_{P_k,i}\| \tag{5.3}$$

$$\|\vec{w}_{t_i}\| = \varrho s_k\|\vec{v}_{P_k,i}\| \tag{5.4}$$

where $\varrho$ is a factor which must satisfy $0 < \varrho < 1$. The significance of $\varrho$ will be explained later in this section. The equivalent vectors formed after apportioning are also shown in the second part of Figure 5.7.

It is found that distributing the escape motion vectors of link $L_i$ merely to the ends $h_i$ and $t_i$ is not enough to solve all problems unless it also shared by the *adjacent* links. In other words the escape motion vectors acting on link $L_i$ must also be apportioned to links $L_{i-1}$ and $L_{i+1}$ (if they exist) for an effective solution.

Figure 5.8 shows the vector at link $L_i$ acting at $P_{k,i}$ being apportioned to the head $h_{i-1}$ of the adjacent link to the left and the tail $t_{i+1}$ of the adjacent one to the right. We call the component vectors $\vec{z}_{h_{i-1}}$ and $\vec{z}_{t_{i+1}}$ respectively. The magnitude

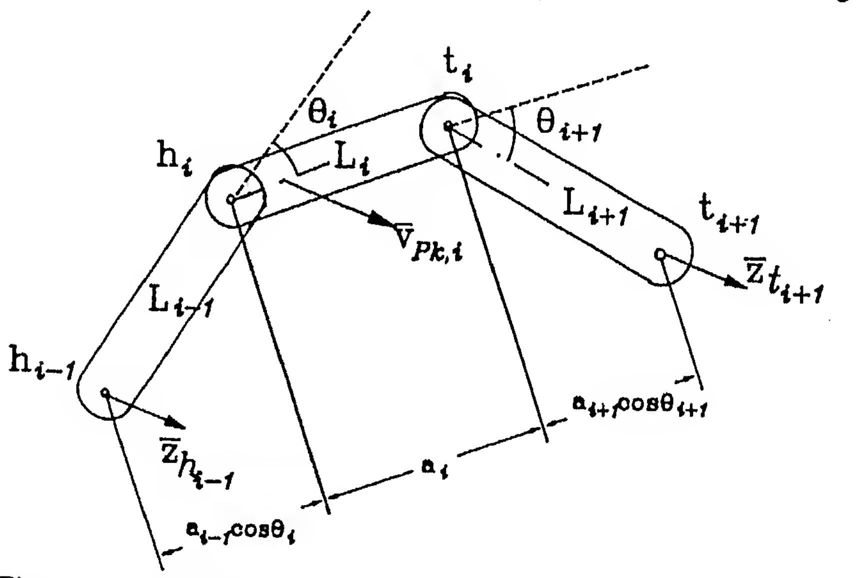


Figure 5.8: Apportioning of escape motion vector to adjacent links

of the component vectors is given by

$$\|\vec{z}_{h_{i-1}}\| = (1 - \varrho)v\|\vec{v}_{P_k,i}\| \tag{5.5}$$

$$\|\vec{z}_{t_{i+1}}\| = (1 - \varrho)(1 - v)\|\vec{v}_{P_k,i}\| \tag{5.6}$$

where

$$v = \frac{(1-s_k)a_i + a_{i+1}\cos\theta_{i+1}}{a_{i+1}\cos\theta_{i+1} + a_i + a_{i-1}\cos\theta_i}$$

and $\varrho$ is the same factor used in Equation 5.3 that assigns a weightage to the apportioning between the same link and adjacent links. A value of $\varrho = 0.75$ means that 75% of the escape motion vector is distributed to the same link while the rest 25% is given to the adjacent links. A value of $\varrho = 0.75$ is found reasonable.

After all apportioning is done the effective motion vectors on the links are acting only at the joints. The effective motion vector acting at joint $i$ is the resultant of all vectors acting there and is represented by $\vec{e_i}$. Figure 5.9 shows the effective motion vectors for the manipulator shown in Figure 5.6.

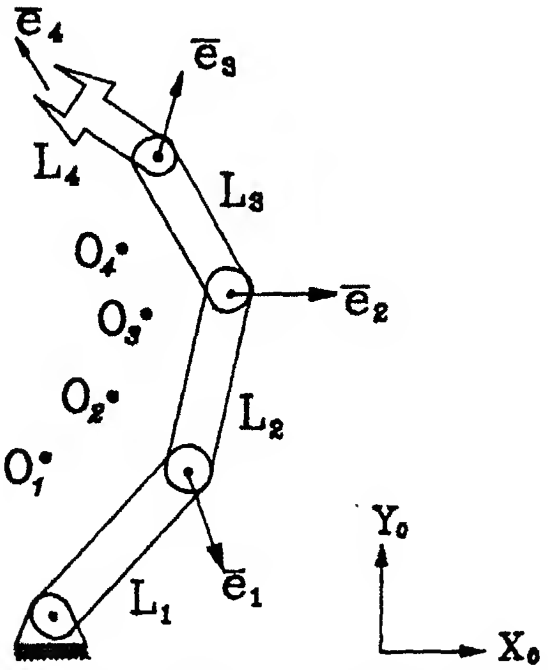


Figure 5.9: Effective motion vectors acting at manipulator joints

### Reorientation of links with static end-effector

The effective motion vectors are essentially directions in which the link joints must move to avoid collision. The kinematic restrictions on the manipulator requires that the manipulator must maintain continuity (i.e. the joints cannot break) and the velocity and acceleration of the moving joints must be within the allowable limits.

The net motion of the manipulator can be thought of as comprising of two components. In the first component of motion, the end-effector moves without any restriction to reach the following point on its trajectory. In the next component, the links of the manipulator reorient themselves *without* moving the end-effector such that collision with nearby objects is avoided. The former component of motion is discussed in Section 5.4.2. In this section we discuss a geometric method of reorienting links based on the effective motion vector acting at the joints.

Figure 5.10 shows a segment of a manipulator where our interest is in link $L_i$ and $L_{i+1}$. Given the position of $h_i$ and $t_{i+1}$ in the coordinate frame $X_0$-$Y_0$ and the

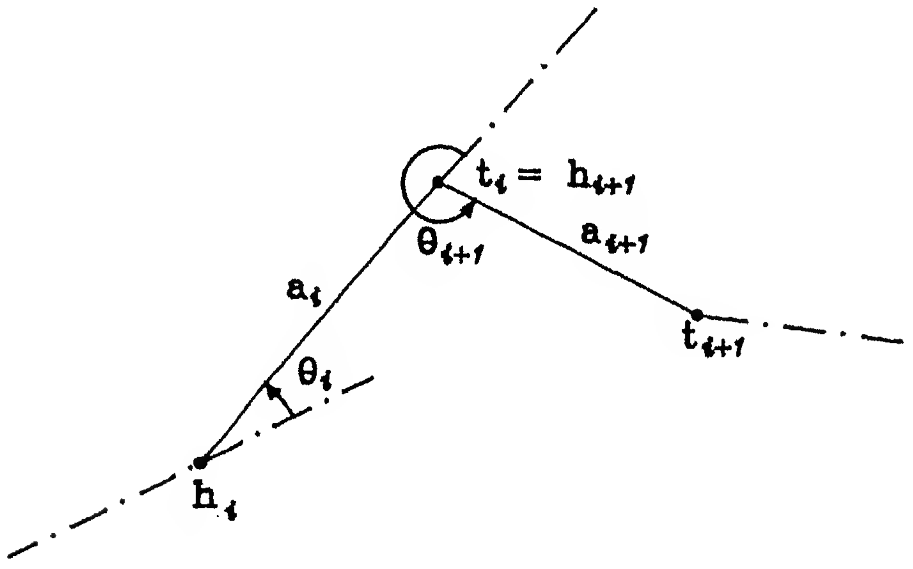


Figure 5.10: Geometry of two links on a plane

link lengths $a_i$ and $a_{i+1}$, the joint angles $\theta_i$ and $\theta_{i+1}$ have to be found. This is a purely geometric problem of fitting two links between its end points and has, in general, only two unique solutions. Let us define two parameters $\Delta x$ and $\Delta y$ as

$$\Delta x \;=\; (t_{i+1})_x - (h_i)_x$$

$$\Delta y = (t_{i+1})_y - (h_i)_y$$

Then by simple trigonometry we may write

$$\begin{aligned} a_i \cos\theta_i + a_{i+1}\cos\theta_{i+1} &= \Delta x \\ a_i \sin\theta_i + a_{i+1}\sin\theta_{i+1} &= \Delta y \end{aligned}$$

or,

$$\Delta x - a_{i+1}\cos\theta_{i+1} = a_i\cos\theta_i \tag{5.7}$$

$$\Delta y - a_{i+1}\sin\theta_{i+1} = a_i\sin\theta_i. \tag{5.8}$$

Eliminating $\theta_i$ from Equation 5.7 we get

$$\Delta x\cos\theta_{i+1} + \Delta y\sin\theta_{i+1} = \frac{(\Delta x)^2 + (\Delta y)^2 + a_{i+1}^2 - a_i^2}{2a_{i+1}} = \mathsf{K} \tag{5.9}$$

Equation 5.9 involves only $\theta_{i+1}$ and can be easily solved. Dividing Equation 5.9 throughout by $\sqrt{(\Delta x)^2 + (\Delta y)^2}$ and assuming

$$\cos\xi = \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} \tag{5.10}$$

$$\sin\xi = \frac{\Delta y}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} \tag{5.11}$$

$$\Rightarrow \qquad \xi = \tan^{-1}\frac{\Delta y}{\Delta x}$$

we may rewrite Equation 5.9 as

$$\cos\xi\cos\theta_{i+1} + \sin\xi\sin\theta_{i+1} = \frac{\mathsf{K}}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} = \mathsf{K}_1 \tag{5.12}$$

i.e.

$$\cos(\xi - \theta_{i+1}) = \mathsf{K}_1$$

which leads to

$$\theta_{i+1} = \xi \pm \cos^{-1}\mathsf{K}_1 \tag{5.13}$$

Putting the value of $\theta_{i+1}$ back into Equation 5.7 we get

$$\sin\theta_i = \frac{\Delta y - a_{i+1}\sin\theta_{i+1}}{a_i} \tag{5.14}$$

$$\cos\theta_i = \frac{\Delta x - a_{i+1}\cos\theta_{i+1}}{a_i} \tag{5.15}$$

which give two corresponding values of $\theta_i$ given by

$$\theta_i = \tan^{-1}\frac{\Delta y - a_{i+1}\sin\theta_{i+1}}{\Delta x - a_{i+1}\cos\theta_{i+1}} \tag{5.16}$$

Since the above method lays down two links $L_i$ and $L_{i+1}$ between its end points $h_i$ and $t_{i+1}$, we call it procedure `LayoutLinks(link[i], link[i+1], head[i], tail[i+1])` . This basic procedure will be used later to describe link fitting methods for redundant manipulators.

Consider Figure 5.11 which shows the effective motion vector $\vec{e}_i$ at the tail of link $L_i$. Let the new required position of the tail of link $L_i$ after a small discrete

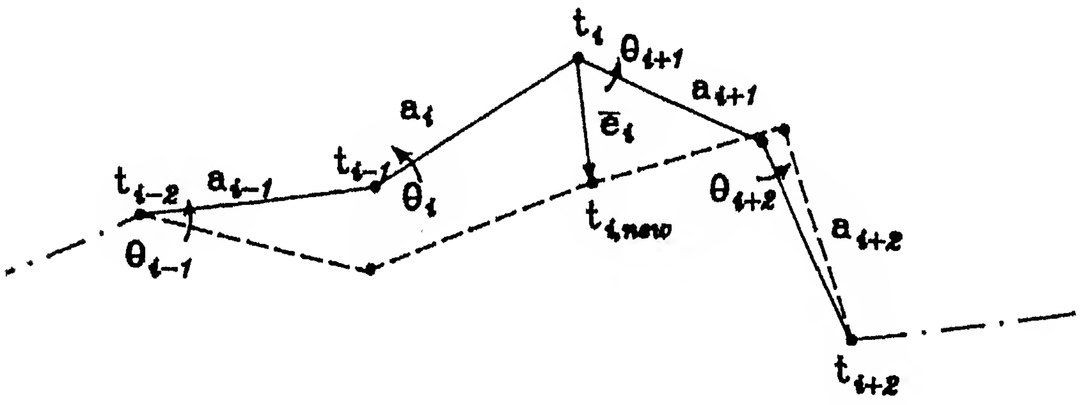


Figure 5.11: Reorienting four links on a plane

step be $t_{i,new}$ (called `newtail[i]` in the algorithm below). It is obvious from Figure 5.11 that the minimum number of links that have to be reconfigured is four viz. two links adjacent to $t_i$ towards the base side and two towards the end-effector side. Therefore, we impose that the movement of $t_i$ be accomplished by appropriately displacing $t_{i-1}$ and $t_{i+1}$, thus calling for the rotation of joints $\theta_{i-1}$, $\theta_i$, $\theta_{i+1}$, $\theta_{i+2}$ and $\theta_{i+3}$. This solution can be algorithmically expressed as

```
procedure MoveTail(link i);
   begin
      LayoutLinks(link[i-1], link[i], head[i-1], newtail[i]);
      LayoutLinks(link[i+1], link[i+2], newtail[i], tail[i+2]);
   end;
```

Algorithm LayoutLinks can be called only when $2 \leq i \leq n-2$ holds, i.e. when there are at least 2 links on *both* sides of $t_i$. However, when $i = 1$ or $i = n-1$, then tail $t_i$ cannot be faithfully brought to $t_{i,new}$. For example, in Figure 5.12, even though the motion vector $\vec{e}_1$ indicates that the tail $t_1$ be brought to $t_{1,new}$, it is not possible to do so because the position of the head of link 1, $h_1$, is immovable. Hence at most, the point $t_1$ can be brought *nearest* to $t_{1,new}$ by rotating the link

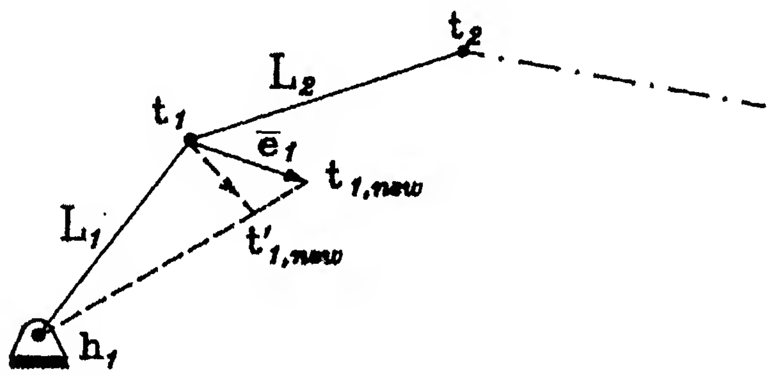


Figure 5.12: Reorienting the first link according to the effective motion vector

about $h_1$. Let the nearest point attained thus by $t_1$ be labelled as $t'_{1,new}$ (called newtailprime[i] in the algorithm below). At the nearest point the slope of line $t_{1,new}h_1$ and line $t'_{1,new}h_1$ is same. Similarly, when tail $t_{n-1}$ has to moved without moving the end-effector ($t_n$), link $L_n$ is rotated about $t_n$ to bring $t_{n-1}$ *nearest* to $t_{n-1,new}$. The algorithm which shifts all links according to the escape motion vectors can thus be expressed in algorithmic fashion as

```
procedure ShiftLink(link i);
   begin
```

```
      if (i = 1) then
         begin
            turn link i about head[i] towards newtail[i]
                     giving newtailprime[i];
            LayoutLinks (link[i+1], link[i+2], newtailprime[i], tail[i+2]);
         end;
      else
      if (i = (n - 1)) then
         begin
            turn link (i+1) about tail[i+1] towards newtail[i]
                     giving newtailprime[i];
            LayoutLinks (link[i-1], link[i], head[i-1], newtailprime[i]);
         end;
      else
         MoveTail(link i);
   end;
```

### Checking motion limits of joints

The joint values, joint velocities and joint accelerations have to be kept within their limiting values when imparting motion to the manipulator. If a joint overshoots its extremity, i.e. when $\theta_i < (\theta_i)_{lower}$ or when $\theta_i > (\theta_i)_{upper}$, the joint value is made equal to the limiting value, i.e. $\theta_i = (\theta_i)_{lower}$ in the former case and $\theta_i = (\theta_i)_{upper}$ in the latter case. This essentially means that joint $i$ is now rigid for movement in a certain direction because link $L_{i-1}$ and link $L_i$ are now locked as one rigid body. The manipulator thus loses its redundancy by one.

When two links lie on a straight line or a joint limit is reached, LayoutLinks is likely to fail if the effective motion vector requires further stretching or bending of the links. This situation is analogous to singularity in the joints. Under such circumstances the redundancy of the manipulator is again exploited. In Figure 5.13 the limiting value of joint $i$ has been reached (i.e. $\theta_i = (\theta_i)_{upper}$) and it is not possible to change $\theta_i$ if the effective motion vector is in the direction shown in the figure.

Since link $L_i$ and link $L_{i-1}$ are locked in this position, `LayoutLinks(link[i-1],`

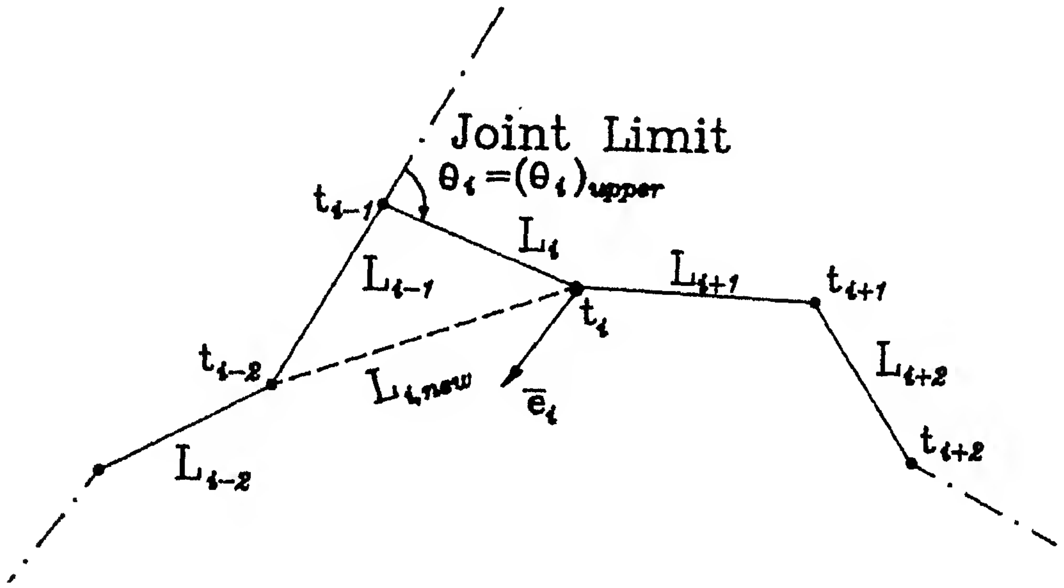


Figure 5.13: Exploiting redundancy at joint limits

`link[i], head[i-1], newtail[i])` fails. Under such circumstances, an imaginary link $L_{i,new}$, (also called `newlink[i]` later) is assumed between $h_{i-1}$ and $t_i$ with link length equal to the current separation between $h_{i-1}$ and $t_i$. The joint value $\theta_i$ is frozen and `LayoutLinks(link[i-2], newlink[i], head[i-2], newtail[i])` is called instead. Similarly, when `LayoutLinks(link[i+1], link[i+2], newtail[i], tail[i+2])` fails, $\theta_{i+2}$ is frozen and `LayoutLinks(newlink[i+1], link[i+3], newtail[i], tail[i+3])` is called with `newlink[i+1]` being the current separation between $t_i$ and $t_{i+2}$.

When joint velocity or acceleration limits are reached the entire manipulator has to be moved at a slower rate to keep the velocity and acceleration values within limits. We reduce the effective motion vector of the affected joints by half and solve again till all joint motion constraints are satisfied. The algorithm for solving the manipulator may thus be stated as follows.

```
procedure ReorientManipulator;
```

```
begin
   memorize all positions of links;
   for i := 1 to n do
       ShiftLink(i);
   for i := 1 to n do
       begin
          Check velocity and acceleration limits of joint i;
          if (limit of joint i is exceeded) then
             begin
                reduce effective motion vector at joint i-1 by half;
                reduce effective motion vector at joint i by half;
                reduce effective motion vector at joint i+1 by half;
             end;
          restore old position of all links;
          ReorientManipulator; (* recursive calling *)
       end;
end;
```

### No restriction on movement of end-effector

Under extremely dangerous situations, the end-effector cannot maintain its position and it must be allowed to move away under the influence of the obstacles around it. In this state, the manipulator behaves as if all its joints are free. All links are moved away from the obstacles depending on the escape motion vector at the joints. In Figure 5.14 the escape motion vectors at each joint are shown. First link $L_1$ is oriented so that $t_1$ is brought nearest to $t_{1,new}$. Let the resulting position be $t'_{1,new}$. Now fixing the position of $t_1$ at $t'_{1,new}$ the next joint at $t_2$ is brought nearest to $t_{2,new}$ as indicated by the escape motion vector $\vec{e}_2$. This process is continued for all links. The solution strategy is expressed in algorithmic form below.

```
procedure SolveFreeManipulator;
```

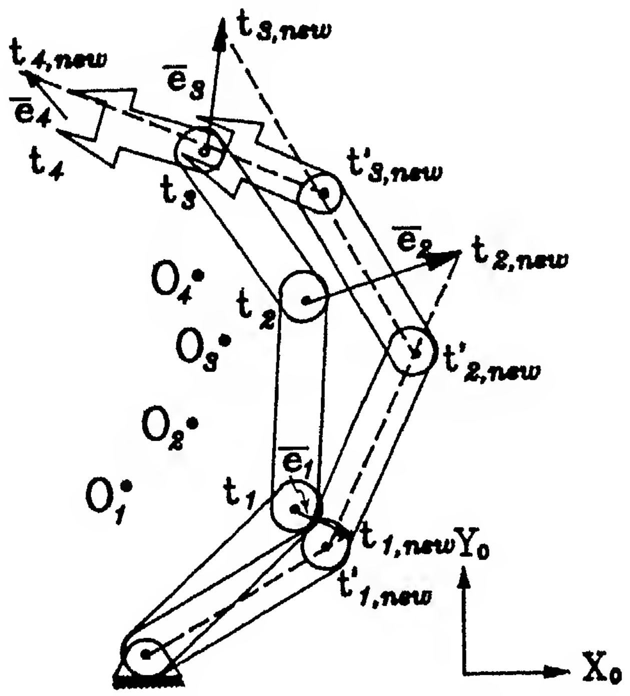


Figure 5.14: Solving a free manipulator

```
begin
   memorize all positions of links;
   newtail[0] := tail[0];
   for i := 1 to n do
       begin
          turn tail[i] about newtail[i-1] towards newtail[i]
                        giving newtailprime[i];
       end;
   for i := 1 to n do
       begin
          Check velocity and acceleration limits of joint i;
          if (limit of joint i is exceeded) then
             begin
                reduce effective motion vector at joint i-1 by half;
                reduce effective motion vector at joint i by half;
                reduce effective motion vector at joint i+1 by half;
              end;
            restore old position of all links;
            SolveFreeManipulator; (* recursive calling *)
          end;
     end;
```

When kinematic joint limits are exceeded, the magnitudes of the escape motion vectors are reduced by half, and the manipulator configuration is solved again.

### 5.4.2 Movement in the absence of obstacles

#### Algebraic solution

When no obstacles are detected near the manipulator the escape motion vectors are absent. The manipulator must now be moved satisfying some other criterion. The best algebraic criterion is that the movement of all joints must be minimum for each incremental movement. Traditionally, this criterion is called the **minimum norm solution** [109]. This problem can be posed as an optimization problem minimizing the sum of squares of all changes in the joint angles. An equivalent method of posing the same problem is to involve the Jacobian matrix and use classical optimization methods.

We now take up a four-link (redundant) manipulator, and derive the equations for the minimum norm solution. It is shown that the problem essentially boils down to solving a set of six linear simultaneous equations. It must be noted that the solution is valid only for small incremental angles.

Figure 5.15 shows a four-link planar manipulator where the each link is represented by a straight line of length $a_i$ for simplicity. The joint angles are denoted by $\theta_1$, $\theta_2$, $\theta_3$ and $\theta_4$ respectively as shown in the figure. The link lengths are represented by $a_1$, $a_2$, $a_3$ and $a_4$ respectively. Then the position of the end effector $(x_E, y_E)$ can be represented as

$$x_E = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

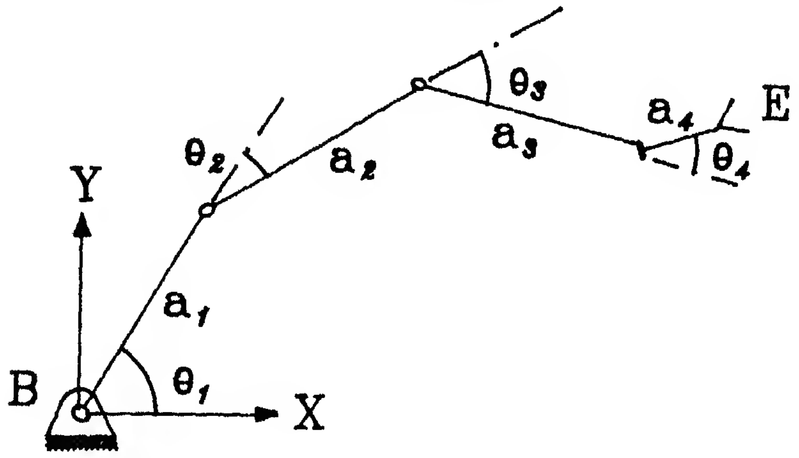


Figure 5.15: Representation of a four link planar manipulator

$$
\begin{aligned}
& +a_4\cos(\theta_1+\theta_2+\theta_3+\theta_4) && (5.17)\\
y_E &= a_1\sin\theta_1 + a_2\sin(\theta_1+\theta_2) + a_3\sin(\theta_1+\theta_2+\theta_3)\\
& +a_4\sin(\theta_1+\theta_2+\theta_3+\theta_4) && (5.18)
\end{aligned}
$$

For small values of change, the relation between the incremental end-effector position and the incremental changes in the joint values may be written as

$$\Delta\vec{x} = J\Delta\vec{\theta} \tag{5.19}$$

where $J$ is the Jacobian matrix. According to Taylor's series expansion we may write the increments in the end-effector position $(\Delta x_E, \Delta y_E)$ in expanded form as

$$
\begin{aligned}
\Delta x_E &= \frac{\partial x_E}{\partial\theta_1}\Delta\theta_1 + \frac{\partial x_E}{\partial\theta_2}\Delta\theta_2 + \frac{\partial x_E}{\partial\theta_3}\Delta\theta_3 + \frac{\partial x_E}{\partial\theta_4}\Delta\theta_4\\
&= J_{11}\Delta\theta_1 + J_{12}\Delta\theta_1 + J_{13}\Delta\theta_3 + J_{14}\Delta\theta_4 && (5.20)\\
\Delta y_E &= \frac{\partial y_E}{\partial\theta_1}\Delta\theta_1 + \frac{\partial y_E}{\partial\theta_2}\Delta\theta_2 + \frac{\partial y_E}{\partial\theta_3}\Delta\theta_3 + \frac{\partial y_E}{\partial\theta_4}\Delta\theta_4\\
&= J_{21}\Delta\theta_1 + J_{22}\Delta\theta_1 + J_{23}\Delta\theta_3 + J_{24}\Delta\theta_4 && (5.21)
\end{aligned}
$$

The Jacobian matrix is a $2\times 4$ matrix whose first column is given as follows.

$$
\begin{aligned}
J_{11} &= \frac{\partial x_E}{\partial\theta_1}\\
&= -a_1\sin\theta_1 - a_2\sin(\theta_1+\theta_2) - a_3\sin(\theta_1+\theta_2+\theta_3)
\end{aligned}
$$

$$-a_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \tag{5.22}$$

$$\begin{aligned} J_{12} &= \frac{\partial x_E}{\partial \theta_2} \\ &= -a_2 \sin(\theta_1 + \theta_2) - a_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ &\quad -a_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.23}$$

$$\begin{aligned} J_{13} &= \frac{\partial x_E}{\partial \theta_3} \\ &= -a_3 \sin(\theta_1 + \theta_2 + \theta_3) - a_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.24}$$

$$\begin{aligned} J_{14} &= \frac{\partial x_E}{\partial \theta_4} \\ &= -a_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.25}$$

The second column of the Jacobian matrix is given by

$$\begin{aligned} J_{21} &= \frac{\partial x_E}{\partial \theta_1} \\ &= a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ &\quad +a_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.26}$$

$$\begin{aligned} J_{22} &= \frac{\partial x_E}{\partial \theta_2} \\ &= a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ &\quad +a_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.27}$$

$$\begin{aligned} J_{23} &= \frac{\partial x_E}{\partial \theta_3} \\ &= a_3 \cos(\theta_1 + \theta_2 + \theta_3) + a_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.28}$$

$$\begin{aligned} J_{24} &= \frac{\partial x_E}{\partial \theta_4} \\ &= a_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \end{aligned} \tag{5.29}$$

The Lagrangian $L$, which serves as the optimization function is now formed as

$$\begin{aligned} L &= (\Delta\theta_1)^2 + (\Delta\theta_2)^2 + (\Delta\theta_3)^2 + (\Delta\theta_4)^2 \\ &\quad +\lambda_1(J_{11}\Delta\theta_1 + J_{12}\Delta\theta_2 + J_{13}\Delta\theta_3 + J_{14}\Delta\theta_4 - \Delta x) \\ &\quad +\lambda_2(J_{21}\Delta\theta_1 + J_{22}\Delta\theta_2 + J_{23}\Delta\theta_3 + J_{24}\Delta\theta_4 - \Delta y) \end{aligned} \tag{5.30}$$

To minimize $L$ we must satisfy the following six conditions.

$$\frac{\partial L}{\partial \lambda_1} = 0 \tag{5.31}$$
$$\frac{\partial L}{\partial \lambda_2} = 0 \tag{5.32}$$
$$\frac{\partial L}{\partial (\Delta\theta_1)} = 0 \tag{5.33}$$
$$\frac{\partial L}{\partial (\Delta\theta_2)} = 0 \tag{5.34}$$
$$\frac{\partial L}{\partial (\Delta\theta_3)} = 0 \tag{5.35}$$
$$\frac{\partial L}{\partial (\Delta\theta_4)} = 0 \tag{5.36}$$

which gives rise to the following six equations.

$$J_{11}\Delta\theta_1 + J_{12}\Delta\theta_2 + J_{13}\Delta\theta_3 + J_{14}\Delta\theta_4 - \Delta x = 0 \tag{5.37}$$
$$J_{21}\Delta\theta_1 + J_{22}\Delta\theta_2 + J_{23}\Delta\theta_3 + J_{24}\Delta\theta_4 - \Delta y = 0 \tag{5.38}$$
$$2\Delta\theta_1 + \lambda_1 J_{11} + \lambda_2 J_{21} = 0 \tag{5.39}$$
$$2\Delta\theta_2 + \lambda_1 J_{12} + \lambda_2 J_{22} = 0 \tag{5.40}$$
$$2\Delta\theta_3 + \lambda_1 J_{13} + \lambda_2 J_{23} = 0 \tag{5.41}$$
$$2\Delta\theta_4 + \lambda_1 J_{14} + \lambda_2 J_{24} = 0 \tag{5.42}$$

In matrix form the above equations can be represented as

$$\begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & 0 & 0 \\ J_{21} & J_{22} & J_{23} & J_{24} & 0 & 0 \\ 2 & 0 & 0 & 0 & J_{11} & J_{21} \\ 0 & 2 & 0 & 0 & J_{12} & J_{22} \\ 0 & 0 & 2 & 0 & J_{13} & J_{23} \\ 0 & 0 & 0 & 2 & J_{14} & J_{24} \end{bmatrix} \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta\theta_3 \\ \Delta\theta_4 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.43}$$

Solution of Equation 5.43 results in the incremental values of the joint angles, $\Delta\theta_1, \ldots, \Delta\theta_4$. These values are added to the old values of $\theta_i$ , $(i = 1,4)$ giving the new values of $\theta_i$ for the next end-effector position.

## 5.5 A Simple Path Planning Scheme

The manipulator, while avoiding obstacles, must fulfill its objective of tracking the prescribed path for path-preference problems and the generated path for point-to-point problems. It was mentioned in Section 5.4.1 that the solution for the joint variables of the manipulator can be divided into two components one of which moves the end-effector without constraints, and the other reorients the links keeping the end-effector static so that collision with nearby obstacles is avoided. Our solution procedure is thus split into the following steps.

**Step 1** : Assuming no collisions, use the method outlined in Section 5.4.2 to move the end-effector by the desired amount.

**Step 2** : If obstacles are encountered, then use the method outlined in Section 5.4.1 to reconfigure the manipulator.

**Step 3** : Superimpose the two solutions to compute the net movement of the manipulator.

If the obstacle environment is such that retaining the end-effector on the assigned path will definitely lead to collision, then the algorithm of Section 5.4.1 is adopted till the danger is over.

In the following sections some examples will be taken to demonstrate the salient features of the algorithm. Results of simulated examples also accompany each description. *For all future reference, it is generally remarked that the simulated examples do not show the point obstacles, but a* **closed polygon** *representing an actual physical obstacle. This is done for identification sake only.* (Section 5.8 describes a simple method to get the point obstacles from the physical obstacle). *Intuitively, it is not wrong to assume that the point obstacles will lie close to the* **vertices** *of the shown polygon. Only the 'visible' vertices of the polygon will have a point obstacle near itself.*

### 5.5.1 Example of motion in an obstacle-free environment

Figure 5.16 shows a manipulator that has to track a path from $E^S$ to $E^G$ shown by the dotted line. The starting configuration and some intermediate configurations of

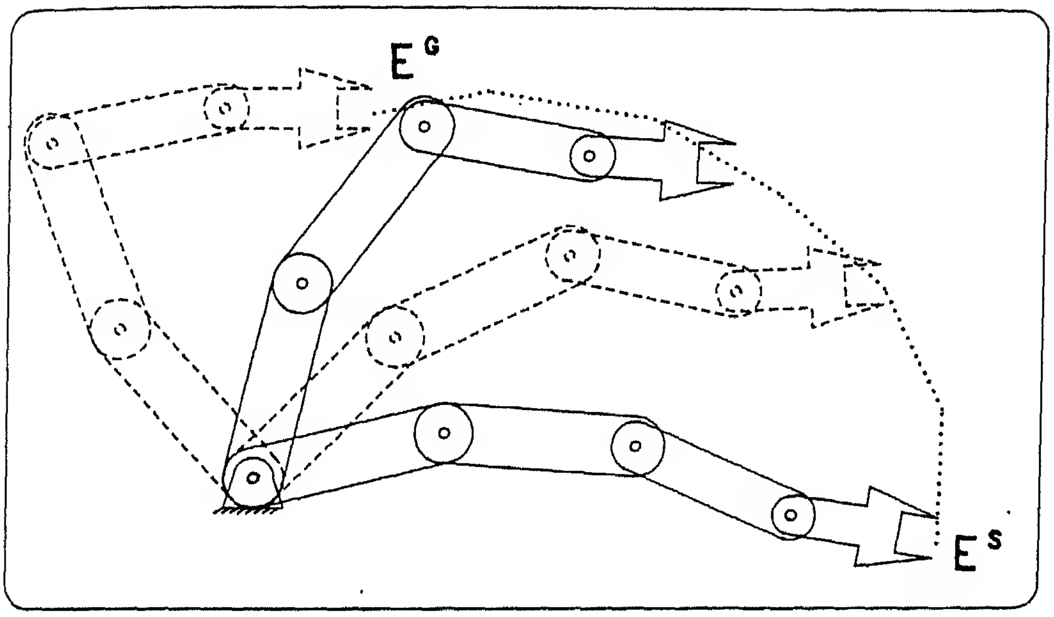


Figure 5.16: Example of Minimum Norm Solution for a four-link manipulator

the manipulator generated by the minimum norm solution are shown in the figure. There is no external obstacle in the workspace and the links do not perceive any collision amongst themselves.

### 5.5.2 Example of motion in an environment with external obstacles

Figure 5.17 illustrates the effect of an external static obstacle on the manipulator when the starting configuration and the end-effector path are same as in Figure 5.16. The intermediate configurations are obtained by superimposing the algorithm of Section 5.4.1 on the minimum norm solution. It is observed from Figure 5.17 that when the end-effector has reached its final position, the configuration attained by the manipulator is considerably different from that attained in Figure 5.16.

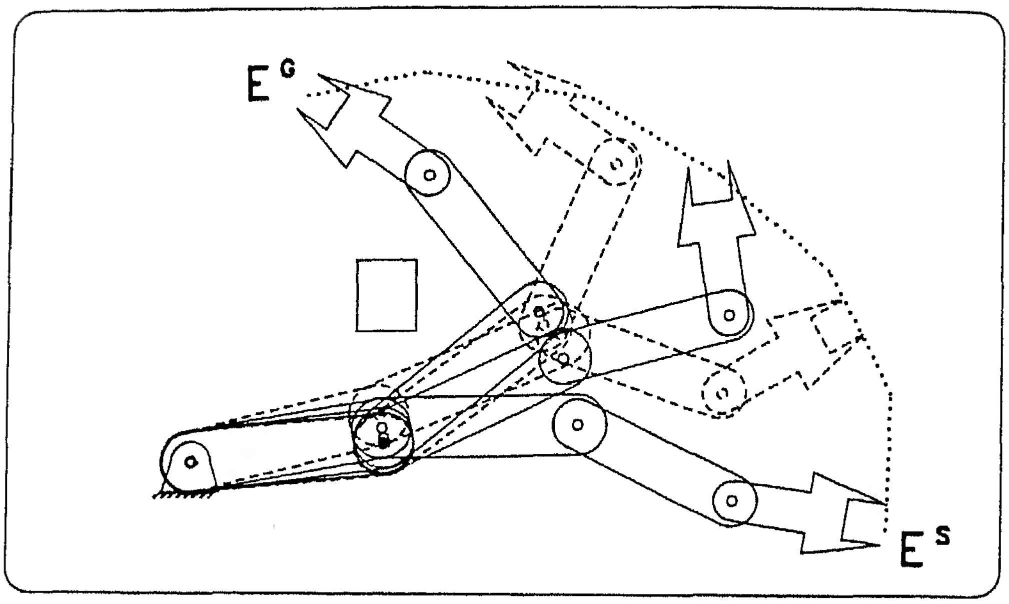


Figure 5.17: Effect of a static obstacle on a four-link manipulator

### 5.5.3 Example of circumventing an obstacle by changing fuzzy rules

The fuzzy rules given in Table 5.1 generate escape vectors which point exactly opposite to the direction of the obstacle. This strategy of escape is quite similar to the artificial potential field approach because it instructs the link to move directly away from the obstacle. The power of using fuzzy logic can be amply demonstrated by a more clever escape strategy. To highlight this point, it is now attempted to solve a slightly difficult problem by manipulating the fuzzy rules. Figure 5.18 shows an obstacle $O$ placed between the end-points of the path $E^S$ and $E^G$ and it is desired to move the end-effector from $E^S$ to $E^G$ with no restrictions on the path, i.e. in point-to-point programming mode. With the rules of Table 5.1 (which repel the manipulator exactly opposite to the obstacle) the problem fails to achieve its objective as shown by the few configurations of Figure 5.18. The end-effector path generated by the program till it fails is shown in Figure 5.19.

The same problem is now solved with a new set of rules that pushes the end-effector towards the base, B of the manipulator. The attraction towards the base

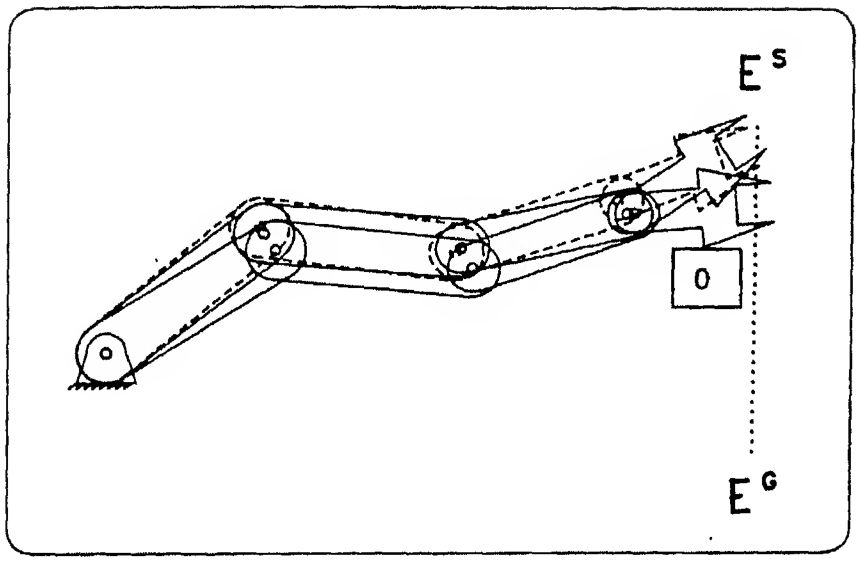


Figure 5.18: A four-link manipulator failing to circumvent an obstacle

is expected to help circumvent the obstacle. The final motion of the end-effector is the resultant influence of the attractive force towards the goal point $E^G$, and the direction suggested by the escape motion vector at the tip of the end-effector, $\vec{e}_n$. The new set of rules are applied only to the end-effector and does *not* require the usual rotation of frames as described in Section 5.3. Rather the link coordinate system is rotated so that the $X_0^R$ axis of the rotated frame aligns itself with the straight line from the base B to the tail of last link $t_n$ (see Figures 5.20 and 5.21). It is once again emphasized for the sake of the cursory reader that these rotations are done in the fuzzy domain and are hence approximate to a resolution of 45°. The rules are formulated in the rotated frame and are given in Table 5.2. The result of the new set of rules is evident from Figure 5.22 which shows a few configurations while the obstacle $O$ is being circumvented. The generated end-effector trajectory is shown in Figure 5.23. This example demonstrates the flexibility of using fuzzy logic for obstacle avoidance. It was seen that the same program exhibits entirely different behaviour if the fuzzy rules are modified. It is possible to change the behaviour of the algorithm entirely by simply manipulating the fuzzy rules which

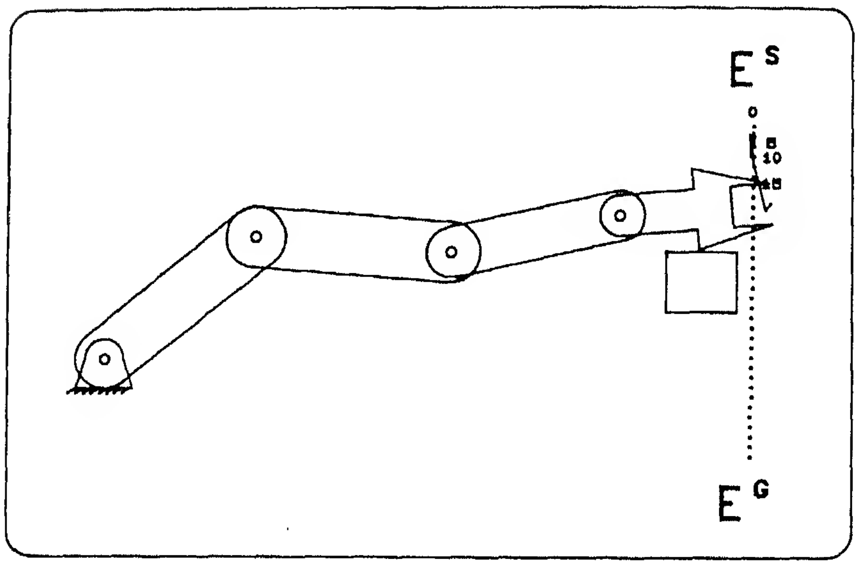


Figure 5.19: End-effector path generated by program for the unsuccessful attempt

govern the escape strategy.

## 5.6 An Improvised Path Planning Scheme

In a dynamic environment amidst unknown obstacles, it is essential to adopt different maneuvering strategies under different situations with varying danger levels. Thus a *label* is attached to each of these situations and the controller of the manipulator is programmed to adopt a suitable path-tracking or path-generating strategy in each case. For example, when the obstacles are perceived to be very close to the manipulator and a collision seems imminent, then the manipulator must surrender its immediate task to achieve the primary objective of collision avoidance. The motion of the obstacles may now push the manipulator in a tight position from where a direct access to its destination is impossible. A suitable maneuvering strategy must now be adopted to circumvent the obstacle if necessary, so that the target position can still be reached. In this section an example is chosen which demonstrates all the features mentioned above, and some results are presented

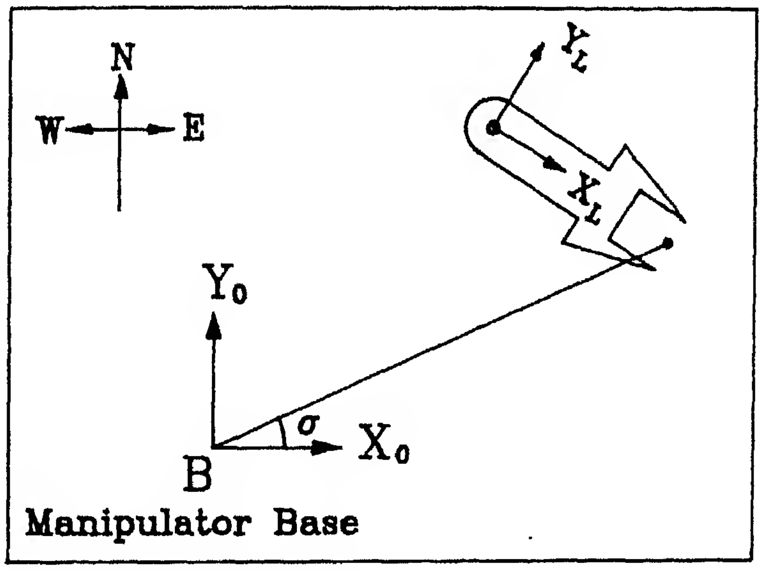


Figure 5.20: Original link coordinate system for the end-effector

that have been obtained from simulation runs.

To describe the concepts involved, a terminology is introduced that has been called the **state of the manipulator.** The *state* of the manipulator indicates its present task status and decides the path-planning strategy to be adopted under the present circumstances. First, all *manipulator states* are briefly enumerated, then an algorithm is presented that describes the strategy adopted for changing between *states*, and finally the path-planning strategy to be adopted in each *state* is described.

### 5.6.1 Brief enumeration of manipulator states

The manipulator is in the *NATURAL* state when it is tracking its assigned path.[2] When the end-effector is stationary, the manipulator is said to be in *PAUSE* state. When the environment and the manipulator both seem to be static, it is a state

[2]In the path-preference problem, the assigned path is the prescribed path. In the point-to-point problem the assigned path is the shortest approach line from the current end-effector position to the goal point.

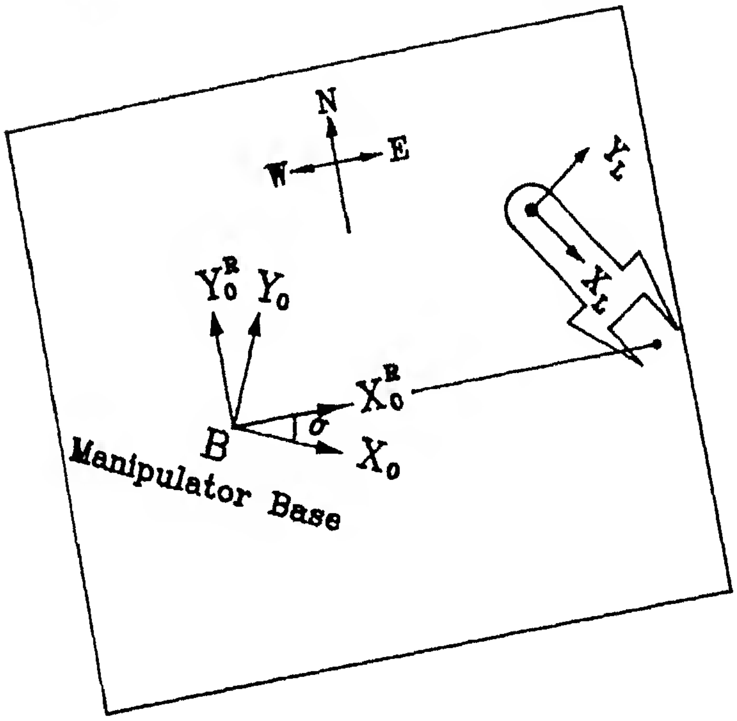


Figure 5.21: Rotated frame for application of rules to the end-effector

of *DEADLOCK*. When the manipulator has completely yielded to its surrounding obstacles, it is in a state of *SURRENDER*. If the manipulator is trying to circumvent an obstacle, it is in a state of *DETOUR*. Finally, when the obstacle has been circumvented, the manipulator approaches its immediate goal, and so, it is in a state of *ADVANCE*.

## 5.6.2 Algorithm for state change of manipulator

For non-trivial problems the *state* of the manipulator will keep changing depending on the status of the environment. The following algorithm outlines the logic for changing the state of the manipulator which permits guidance of the end-effector through a collision-free track. In the algorithm below, the *states* of the manipulator are maintained on a stack. When the manipulator has reached the *NATURAL* state after a successful detour, the stack pointer is reset.

```
procedure DecideManipulatorState;
```

| | IF | AND | THEN |
|---|---|---|---|
| *Rule* | *Obstacle Distance is* | *Obstacle Angle is* | *Move Observation Post* |
| *1* | Near | North | West |
| *2* | Near | South | West |
| *3* | Near | North-East | West |
| *4* | Near | North-West | West |
| *5* | Near | South-East | West |
| *6* | Near | South-West | West |
| *7* | Near | East | West |

Table 5.2: Set of rules for pulling the end-effector towards the base

```
begin
   if ((State = NATURAL) and (goal is reached)) then exit;
   if (State = NATURAL) and
                ((EE path is not free) or (danger is high)) then
       begin
          push(State);
          State := PAUSE;
       end;
   if (State = PAUSE) then
       begin
          if (danger level is low) and (EE path is free) then
              pop(State);
          else
             if (danger is increasing) then
                 begin
                    State := SURRENDER;
                    AbandonPoint := current end-effector location;
                 end;
       end;
```

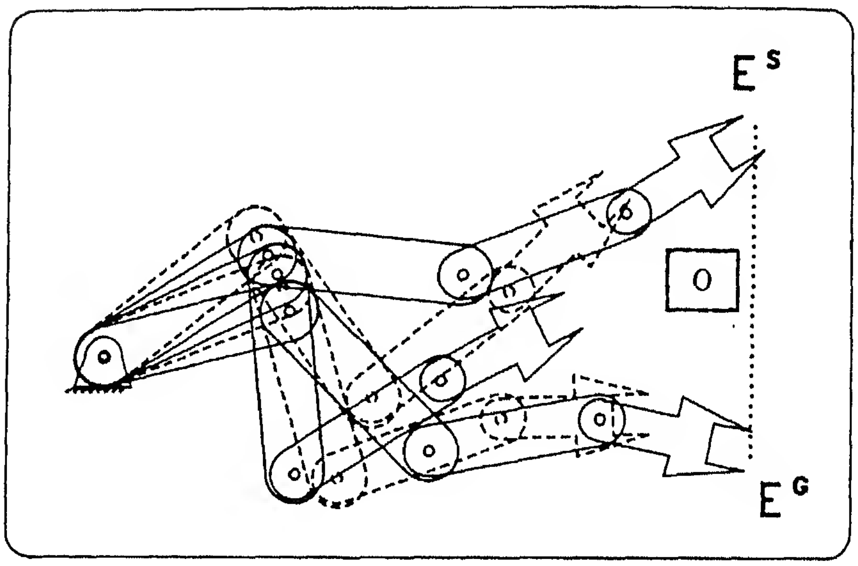


Figure 5.22: A four-link manipulator circumventing a static obstacle by appropriately framing fuzzy rules

```
if ((State = PAUSE) or (State = SURRENDER)) then
    if (the sensor readings have not changed for long) then
        State := DEADLOCK;
if (State = DEADLOCK) then
    State := DETOUR;
if (State = DETOUR) or (State = ADVANCE))
          and (danger is high) then
    begin
       push(State);
       State := PAUSE;
    end;
if (State = DETOUR) and (magnitude of escape vector is small) then
    begin
       push(State);
       State := ADVANCE;
```

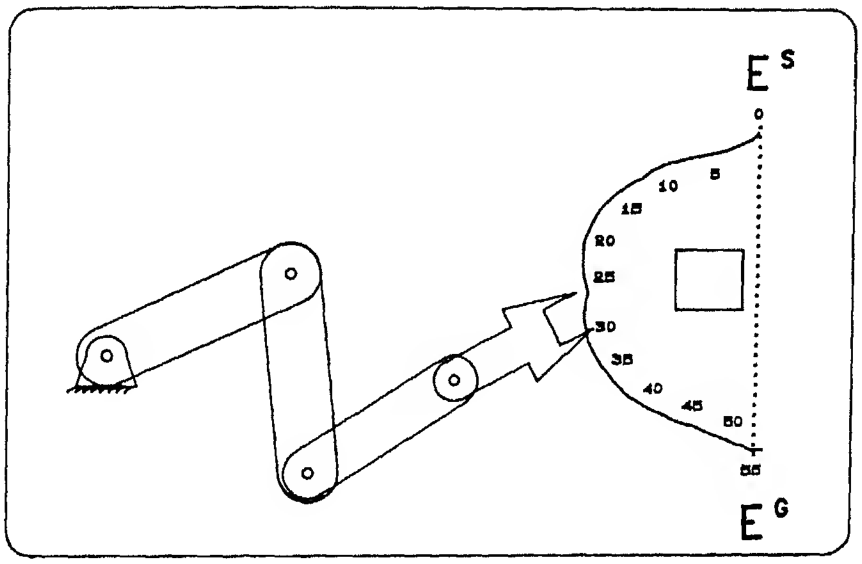


Figure 5.23: End-effector trajectory generated by the program for circumventing a static obstacle

```
      end;
   if (State = ADVANCE) then
      begin
         if (point-to-point) then
            if (goal reached) then exit;
         if (path preference) then
            if (end-effector has reached AbandonPoint) then
               begin
                  State := NATURAL;
                  reset stack pointer;
               end;
      end; (* of State = ADVANCE *)
end; (* of procedure DecideManipulatorState; *)
```

### 5.6.3 Example of path planning strategies dependent upon manipulator states

Our objective to define a manipulator state is to formalize different situations faced by the manipulator, under which different strategies of end-effector path-planning should be employed. In the presence of moving obstacles it may sometimes be fatal to pursue a path when collision seems imminent. In this situation, the manipulator is made to drift along with the obstacles to save an impending collision. This might, at times, land the manipulator into a tight corner from where recovery might be difficult. For instance, the obstacles can push the manipulator away till it is very far from its goal. One of the obstacles may then stop suddenly, preventing the manipulator from directly accessing its normal route. The usage of different *states* and the associated path-planning strategies are illustrated in the following sections through the help of an example of Figure 5.24 where the end-effector is to traverse a path from $E^S$ to $E^G$ and an obstacle is simultaneously moving towards left.

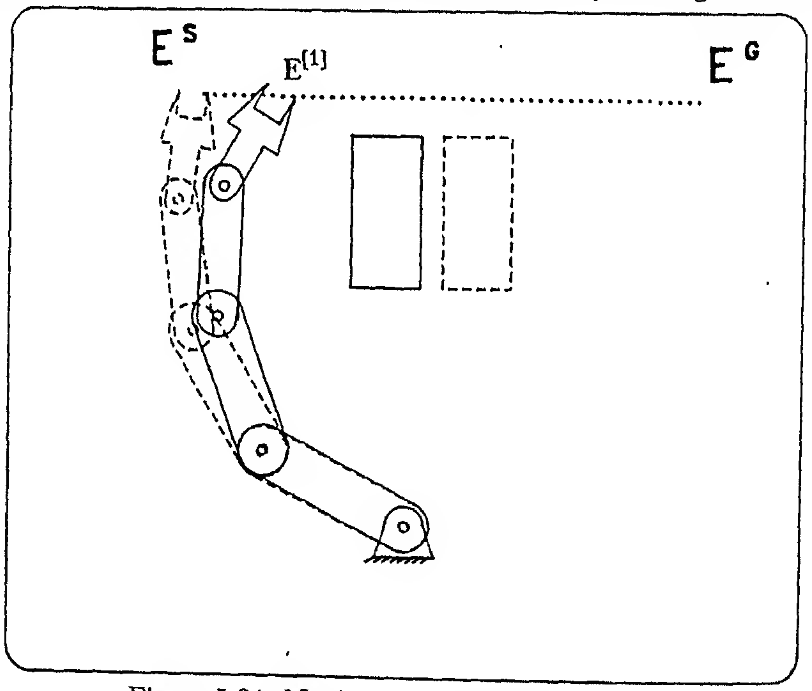


Figure 5.24: Manipulator in *NATURAL* state

### NATURAL state

In the *NATURAL* state, the end-effector is tracking the assigned path in the path-preference problem, or is on the shortest path joining $E^S$ and $E^G$ in the point-to-point programming problem. In this state the end-effector is able to move ahead on the assigned path while simultaneously the links can adjust or reorient to avoid collision. This movement is also depicted in Figure 5.24, where the path from $E^S$ to $E^{[1]}$ is followed in the *NATURAL* state.

### PAUSE state

While the manipulator is tracking a path, the presence of an obstacle *on* the end-effector path or very near any other link will take the manipulator to the *PAUSE* state. In this state the end-effector remains stationary while the links reorient themselves using the strategy outlined in Section 5.4.1 to avoid a collision. In Figure 5.25, as the obstacle moves towards the manipulator, sensor readings indicate high danger of collision. Thus for the duration of obstacle movement from $O^{[1]}$ to $O^{[2]}$ the end-effector remains stationary and the manipulator reconfigures itself from the dotted line configuration to the firm line one.

### SURRENDER state

When the manipulator is in a *PAUSE* state and the possibility of collision keeps increasing, the manipulator must yield to the movement of the obstacle. The manipulator reaches a state of *SURRENDER* and moves away from the approaching obstacle by the method outlined in Section 5.4.1. In this state, the path tracking by end-effector is totally abandoned and collision avoidance becomes the only objective. The dotted line configuration in Figure 5.26 shows the final position attained by the manipulator after it has been dislodged from its intended path and pushed aside from $E^{[1]}$ to another position $E^{[2]}$ beyond $E^S$ as shown in the figure. At this instant of time, the obstacle O has stabilized at the position shown by the dotted line in Figure 5.26.

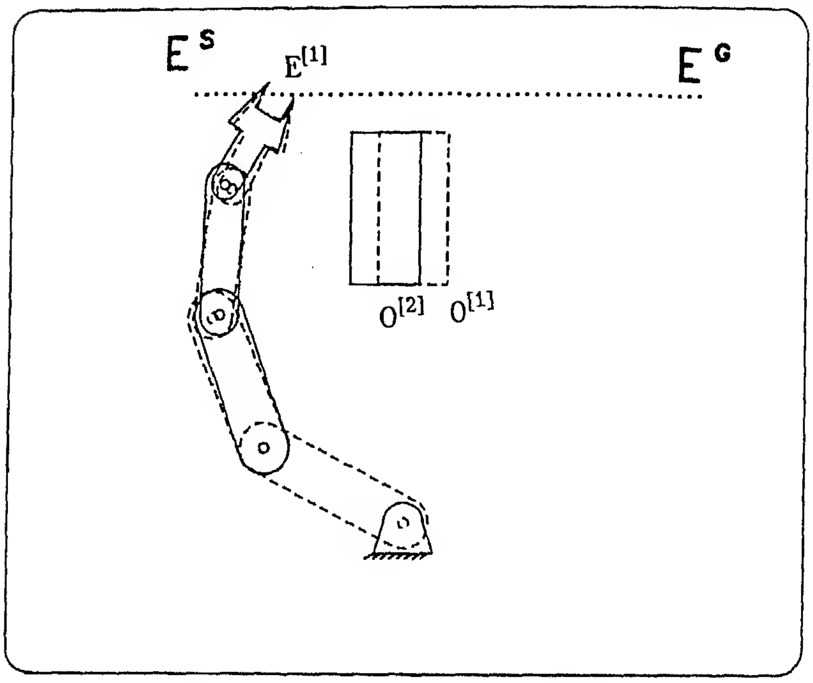


Figure 5.25: Manipulator in *PAUSE* state

## DEADLOCK state

When the manipulator has been static for quite some time at a certain position, it has reached a state of *DEADLOCK* meaning thereby that the sensor readings and hence the obstacles have been stationary for some time. A *DEADLOCK* state signals the manipulator to attempt a recovery by possibly circumventing an obstacle. At the position shown by the dotted line in Figure 5.26, the manipulator is in a state of *DEADLOCK*.

## DETOUR state

In the *DETOUR* state, a new path of the end-effector is planned to restore the manipulator to a favorable position from where it might achieve its goal easily. This will then require circumventing the obstacle. The path-planning of the end-effector in this state is based on the escape vector at the end-effector viz. the vector $\vec{e}_n$. An attempt is made to retract the end-effector towards the base of the manipulator. This is achieved by applying the fuzzy rules of Table 5.2 to deter-

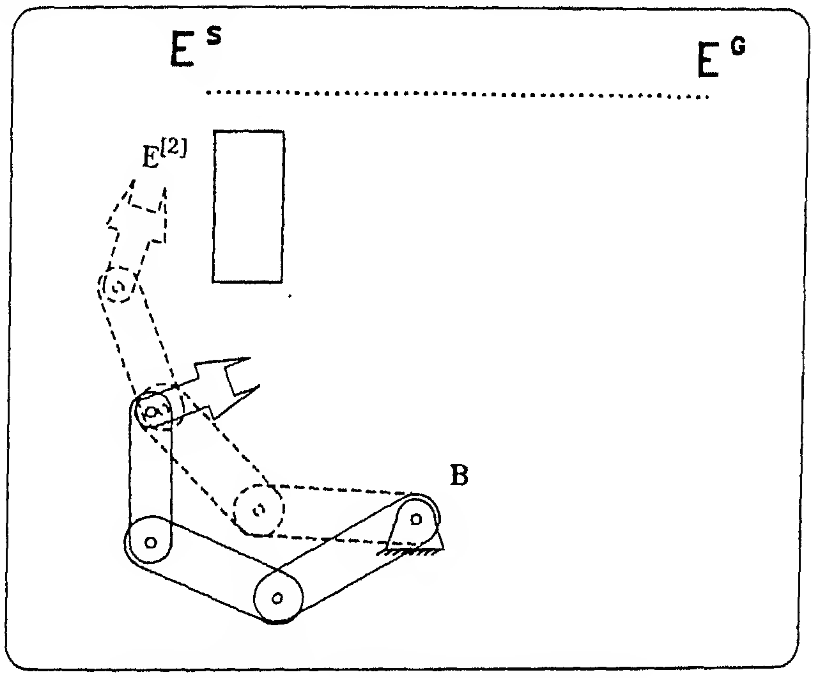


Figure 5.26: Manipulator in *DETOUR* state

mine the appropriate escape vector. The net motion imparted to the end-effector for each incremental movement, is the resultant of this vector plus a component towards the goal $E^G$. When the magnitude of the escape motion vector falls below a threshold value, the *DETOUR* procedure should end and allow the manipulator to pass on to the next state. The firm line configuration in Figure 5.26 shows a position of the manipulator in the *DETOUR* state. At the end-effector position shown by $E^{[3]}$ in Figure 5.27, the sensors indicate that the escape motion vector is small, and thus the *DETOUR* process should end here.

### ADVANCE state

This state follows immediately after a *DETOUR* operation has succeeded. There are two possibilities to guide the end-effector depending on whether the problem is posed as a point-to-point programming problem or a path-preference problem.

In the point-to-point programming problem, the *ADVANCE* procedure must lead the end-effector towards the goal $E^G$ from its current position. The incre-

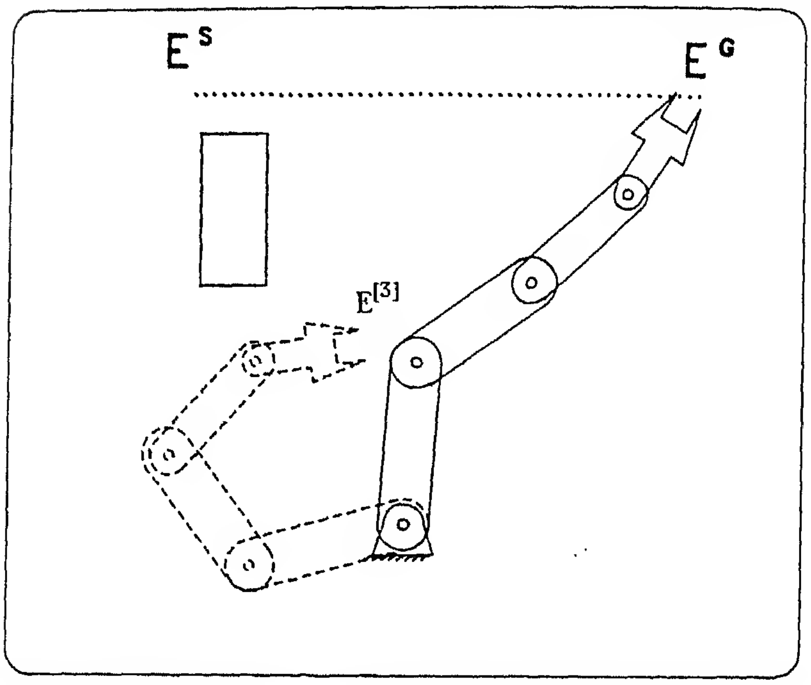


Figure 5.27: Manipulator in *ADVANCE* state for point-to-point problem

mental steps required for moving the end-effector are calculated at each stage from the current location of the end-effector by infinitesimally advancing it towards the goal. In this problem, the manipulator can never intentionally reach the *NATURAL* state. The firm line configuration in Figure 5.27 shows the final position attained by the manipulator while advancing towards the goal in the point-to-point programming problem.

In the path-preference problem the end-effector attempts to head towards that point on the prescribed path from where it was compelled to leave it. This point was shown as $E^{[1]}$ in Figure 5.25. When the point has been reaccessed, the manipulator is hopefully in a different configuration with the obstacle on its other side. This is illustrated in Figure 5.28 where the same point $E^{[1]}$ has been accessed through a configuration (represented by the dotted line) which is different from the one shown in Figure 5.25. At this instant, the manipulator goes into the *NATURAL* state and resumes journey from $E^{[1]}$ to $E^G$ as shown by the firm line configuration in Figure 5.28.

Figure 5.29 gives the complete path generated by the algorithm for a point-

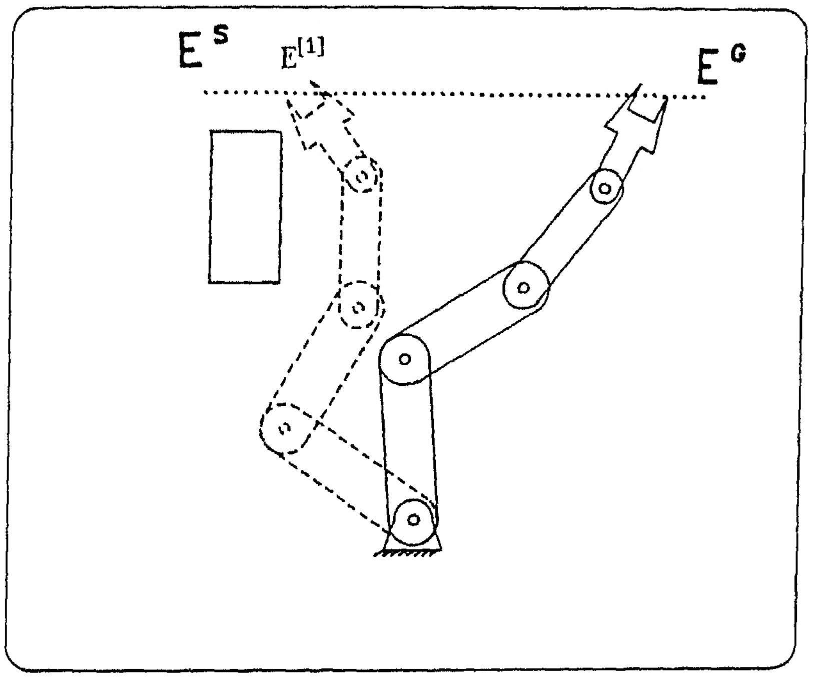


Figure 5.28: Manipulator in *ADVANCE* state in path-preference problem

to-point programming problem. The numbers written beside the path show the progress of the manipulator. The two extreme positions of the obstacle are also shown in the figure. Figure 5.30 shows the complete trajectory of the manipulator for the path-preference problem. The program for implementing this algorithm is written in C and works in the DOS environment on an IBM-AT working at 10MHz. With eight fuzzy rules, each cycle takes about **2.43 seconds** for the four-link manipulator shown.

Figures 5.31 and 5.32 show the joint-value and joint-velocity plots of the manipulator for the same problem in the point-to-point programming case. The assumed joint limits are $-60°$ and $120°$ for $\theta_1$, $-120°$ and $120°$ for $\theta_2$, $-90°$ and $90°$ for $\theta_3$ and $-100°$ and $100°$ for $\theta_4$. The joint-velocity and joint-acceleration limits have been put as $5°sec^{-1}$ and $5°sec^{-2}$ respectively for all joints. For this example the run time was around 300 seconds for the point-to-point programming problem and about 500 seconds for the path-preference problem.

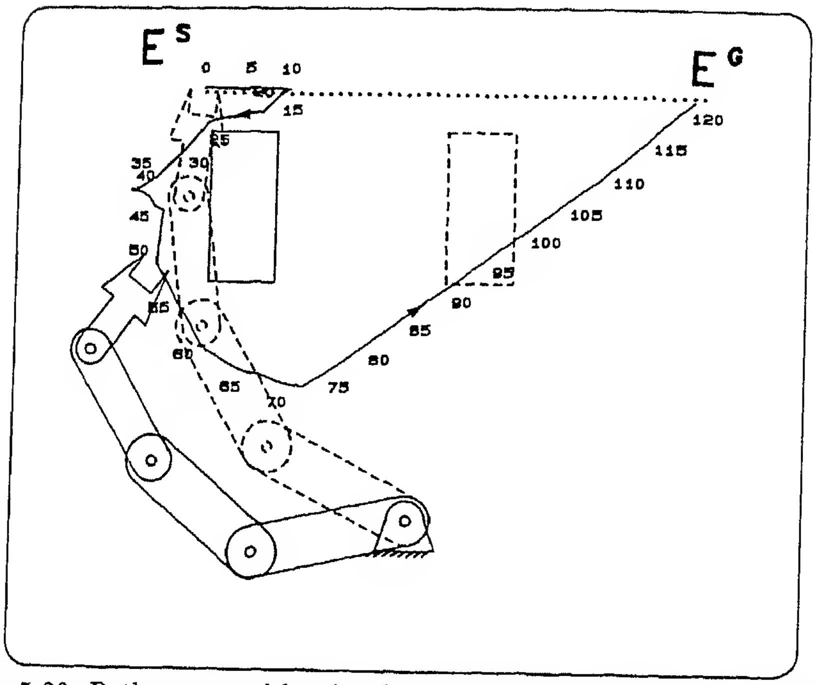


Figure 5.29: Path generated by the algorithm for the point-to-point problem

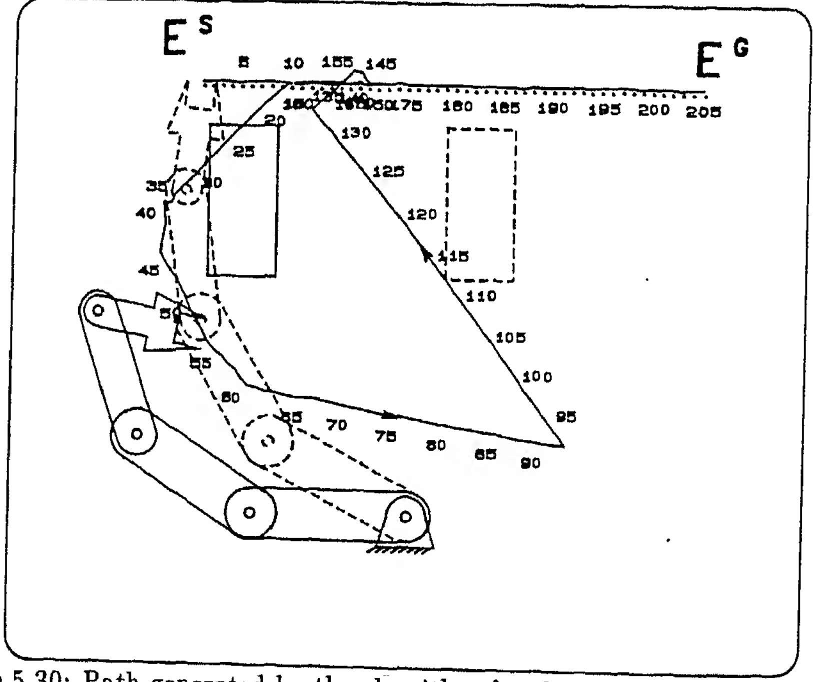


Figure 5.30: Path generated by the algorithm for the path-preference problem

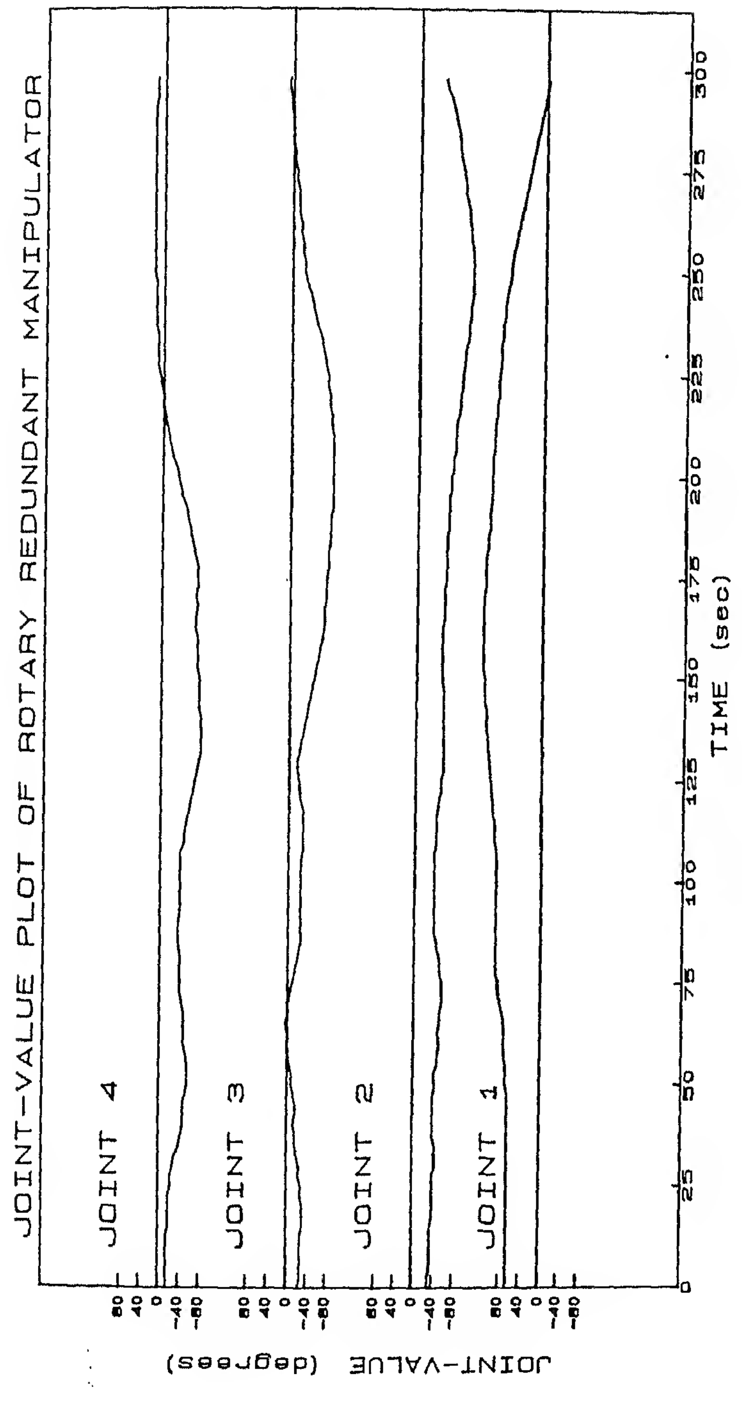


Figure 5.31: Joint-value plot of four-link rotary redundant manipulator

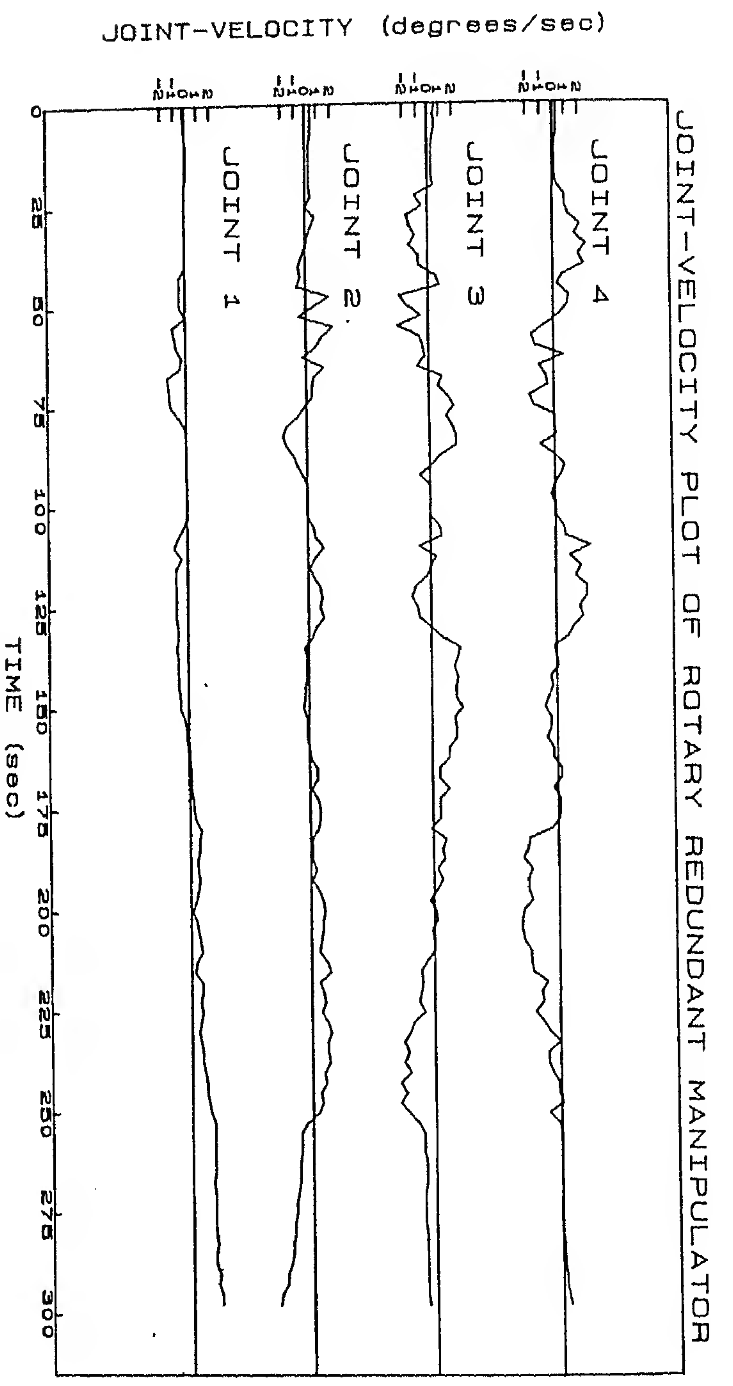


Figure 5.32: Joint-velocity plot of four-link rotary redundant manipulator

### 5.6.4 The algorithm for motion planning cycle

A discussion about path-planning will be incomplete if the correct *sequence* of steps is not mentioned. This sequence is important for a proper coordination of the motion of the manipulator amongst moving obstacles. Just as mentioned in Section 4.4.2, we work with a predicted positions of the manipulator and the point obstacles in the perceived world model to add an element of safety into the analysis. So, instead of working with the *present* position of the point obstacles one must work with the *predicted* position of the point obstacles. The joint values are easily ascertained by readings from internal sensors. This defines the position of the manipulator precisely. At the same time the next end-effector location is also known if the manipulator does not change its course abruptly. The predicted position of the manipulator is found by applying the obstacle-free incremental joint solution of Section 5.4.2 to the current position of the manipulator. All distance and angle calculations (of Section 5.3) are done on the basis of the predicted position of the point obstacles and the manipulator.

The algorithm which describes the motion planning cycle is as follows.

```
procedure RunManipulator;
   begin
      repeat
         take sensor readings and build the Perceived World Model;
         find predicted position of the point obstacles and manipulator;
         DecideManipulatorState;
         find next position of manipulator depending on state;
         move the end-effector to its next decided position;
      until goal is reached;
   end;
```

Finding the predicted position of the point obstacles in the perceived world model is an error-prone job because of proper obstacle identification. When the obstacles are moving at a high speed, it is possible that an obstacle at one instant will appear to vanish the next instant. There is no easy remedy for this problem but to increase the sampling rate. In the simulation runs, it is assumed for simplicity that these awkward situations do not arise and the program will correctly identify the same obstacle if the separation between the two consecutive positions of the point obstacles is not large.

### 5.6.5 Complexity of the motion planning algorithm

Solving the manipulator configuration using the minimum norm method involves a set of $n+2$ ($n$ = number of links) linear simultaneous equations. It is thus an $\mathcal{O}(n)$ operation. Finding the manipulator state involves analysis of sensory readings to find the perceived danger level through fuzzy logic. The compositional rule of inference involves ($\mathrm{N}_d\ \mathrm{N}_a^2$) comparisons for every rule that is processed (This is true only for the formulation strategy presented in this chapter, not in general). The reorientation of links by geometric fitting is a linear operation in the worst case because each link will be considered once i.e. $\mathcal{O}(n)$ complexity. When joint velocity and acceleration limits are reached, the escape vectors are adjusted and the algorithm is invoked again with the new values. An upper bound has been put on the number of recursive calls allowed for this procedure and thus, in the worst case it will take $p\ \mathcal{O}(n)$ time, where $p$ is a constant determining the upper bound on the number of recursive calls. The overall complexity of the algorithm for $r_n$ rules is thus $p\ \mathcal{O}(n) + r_n(\mathrm{N}_d\ \mathrm{N}_a^2)$.

## 5.7 A Scheme for Sensor Mounting

### 5.7.1 Basic criterion for placement of sensors

The preceding discussion on collision avoidance strategies had assumed that the point obstacles $O_j$ are already provided as inputs. Determining the point obstacles in the perceived world model from actual sensory data in the real world is highly sensor dependent. This chapter concludes by providing a suggested scheme of sensor placement on the links and a suggestion for building the point obstacles in the perceived world model.

It is assumed that sensors resembling ultrasonic sensors are mounted on the body of the links to gather data about the environment. It is thus enough to know the proximity of the obstacle from each link to work with the collision avoidance algorithm. The shape and size of the obstacles is immaterial in our discussion as long as the obstacles are large enough to be detected by the sensors.

One of our primary concerns while mounting sensors is to minimize the shadow area. Figure 5.33 shows a possible scheme of mounting sensors with four sensors on each link, one pair guarding each side of a link. Although the sensors can in

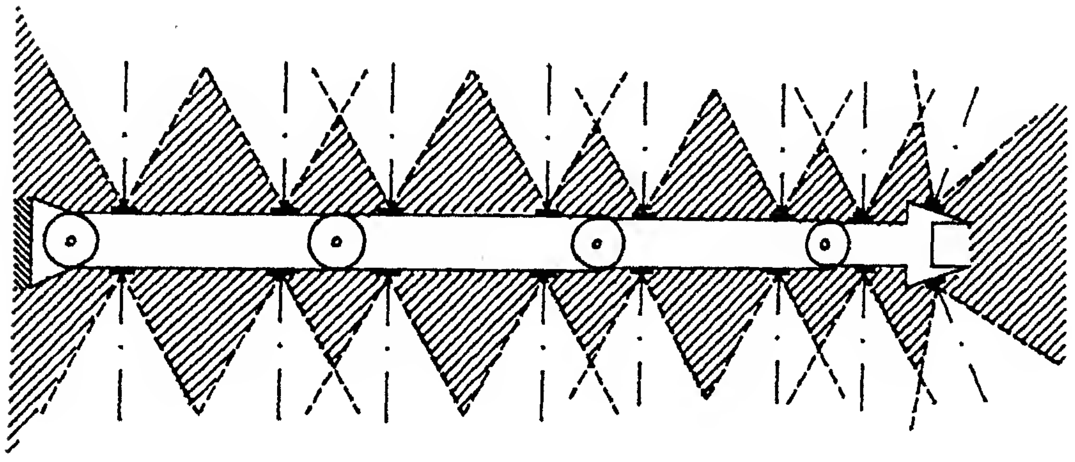


Figure 5.33: A tentative scheme for mounting sensors

effect "see" the entire side of the link, they are not very effective beyond a beam angle of 30° as discussed in Section 3.2. For the sake of illustration, we assume that the sensor cannot detect anything beyond ±30°, and the area which lies beyond this range falls under the sonic shadow. The shadow for this scheme when all joint angles are zero, is also shown in Figure 5.33 with shaded lines. Figure 5.34 shows the shadow region for the same manipulator in a different configuration. It is observed that large areas lying close to the links (consider e.g. top side of link 2) fall inside the shadow region. These areas are thus not guarded by the sensors, and any obstacle which creeps inside this region is likely to hit the manipulator and cause damage. A rigorous analysis is now presented for deciding the best position for mounting the sensors along the length of the link.

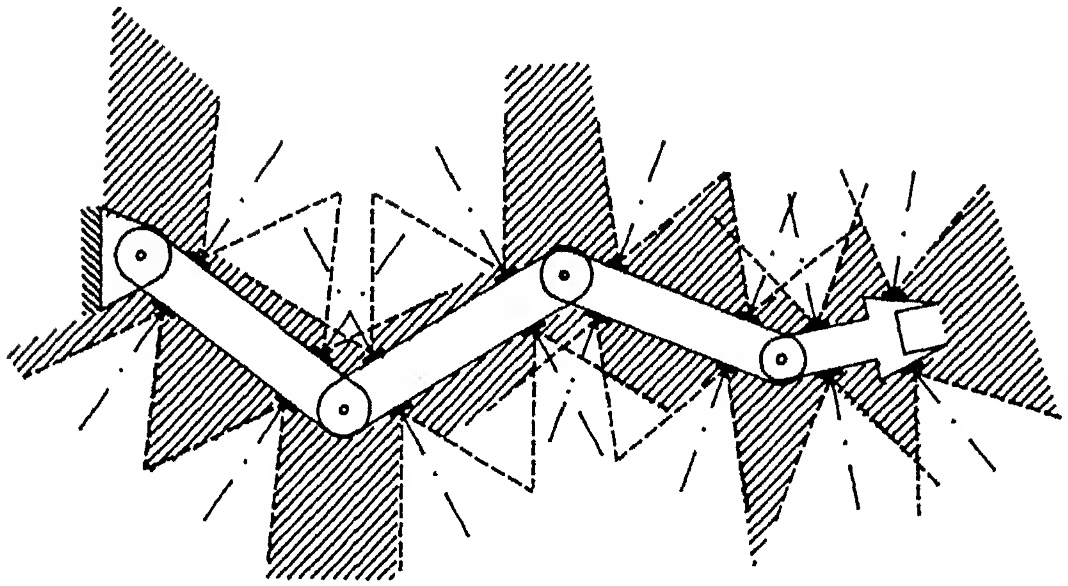


Figure 5.34: Shadow areas for proposed scheme

### 5.7.2 Optimal placement of sensors on a link

The objective in this section is to decide upon an optimal scheme of sensor placement on each link. This problem is posed as a minimization problem whose objective function is the *shadow area.* The result of the minimization determines two numbers $s_1$ and $s_2$ which decide the proportion along the link length at which the sensors have to be placed. Note that the parameters $s_1$ and $s_2$ are the same ones introduced in Section 5.3.

Figure 5.35 shows four sensors mounted on a link with two on each side. Let the first sensor be mounted at a distance $s_1 a_i$ from the head $h_i$ of link $L_i$. The other sensor is mounted at a distance of $s_2 a_i$ from $h_i$. We consider a two-link manipulator to frame the objective function. Figure 5.36 shows a two-link manipulator where the width of the links $b_i$ has been put to zero for clarity. We consider an effective half-beam angle $\alpha = 30°$ and assume that the beam does not affect regions that lie too far away from the source. Thus intersection of shadow regions of only *adjacent* beams are considered in the following analysis. The area of the shadow regions is found from Figure 5.36 to be

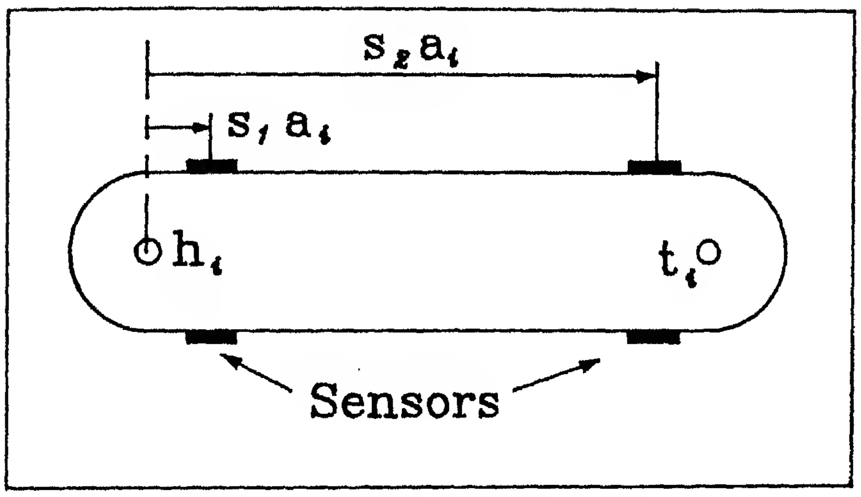


Figure 5.35: Placement of sensors on a link

$$\Delta_{shadow}(\theta_2) = 2\Delta_1 + 2\Delta_2 + \Delta_3 + \Delta_4 + 2\Delta_5 + 2\Delta_6 \tag{5.44}$$

We now derive the areas of the individual areas ($\Delta$s) with reference to Figure 5.36. The area of $\Delta_1$, $\Delta_2$, $\Delta_5$ and $\Delta_6$ is given by

$$\Delta_1 = \frac{(s_1 a_1)^2}{2\tan\alpha} \tag{5.45}$$

$$\Delta_2 = \frac{a_1^2(s_2 - s_1)^2}{4\tan\alpha} \tag{5.46}$$

$$\Delta_5 = \frac{a_2^2(s_2 - s_1)^2}{4\tan\alpha} \tag{5.47}$$

$$\Delta_6 = \frac{a_2^2(1 - s_2)^2}{2\tan\alpha} \tag{5.48}$$

The area $\Delta_3$ and $\Delta_4$ is a function of the joint value $\theta_2$ and is derived using coordinate geometry. Figure 5.37 shows a quadrilateral $Q_1Q_2Q_4Q_3$ formed by the two links $L_1$ and $L_2$ and the edges of the two beams on the links. Let $Q_1$ be the origin $(0,0)$. Then the point $Q_2$ is given by

$$Q_2 \equiv (a_1(1 - s_2), 0)$$

and $Q_3$ by

$$Q_3 \equiv (-a_2 s_1 \cos\theta_2, a_2 s_1 \sin\theta_2).$$

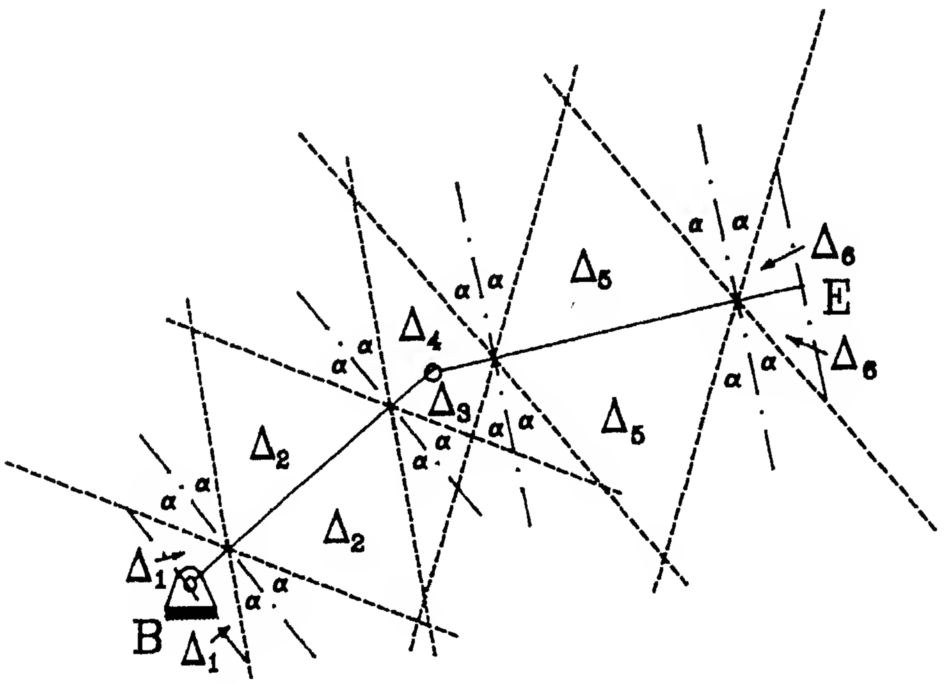


Figure 5.36: Shadow regions for a two-link manipulator

The point $Q_4$ is the intersection of lines $Q_2Q_4$ and $Q_3Q_4$. The equation of line $Q_2Q_4$ is given by

$$y = \frac{a_1(1-s_2)-x}{\tan\alpha} \tag{5.49}$$

while the equation of line $Q_3Q_4$ is

$$y = \frac{x + a_2 s_1 \cos\theta_2}{\tan(\theta_2+\alpha)} + a_2 s_1 \sin\theta_2. \tag{5.50}$$

Solving Equations 5.49 and 5.50 yields

$$\begin{aligned} x_{Q_4} &= \frac{a_1(1-s_2)\cot\alpha - a_2 s_1 \cos\theta_2 \cot(\theta_2+\alpha) - a_2 s_1 \sin\theta_2}{\cot(\theta_2+\alpha)+\cot\alpha} \\ y_{Q_4} &= \frac{a_1(1-s_2)-x_{Q_4}}{\tan\alpha} \end{aligned} \tag{5.51}$$

Then the area of the quadrilateral $Q_1Q_2Q_4Q_3$ is found from

$$\Delta_3 = \sqrt{s(s-q_1)(s-q_2)(s-q_3)(s-q_4)} \tag{5.52}$$

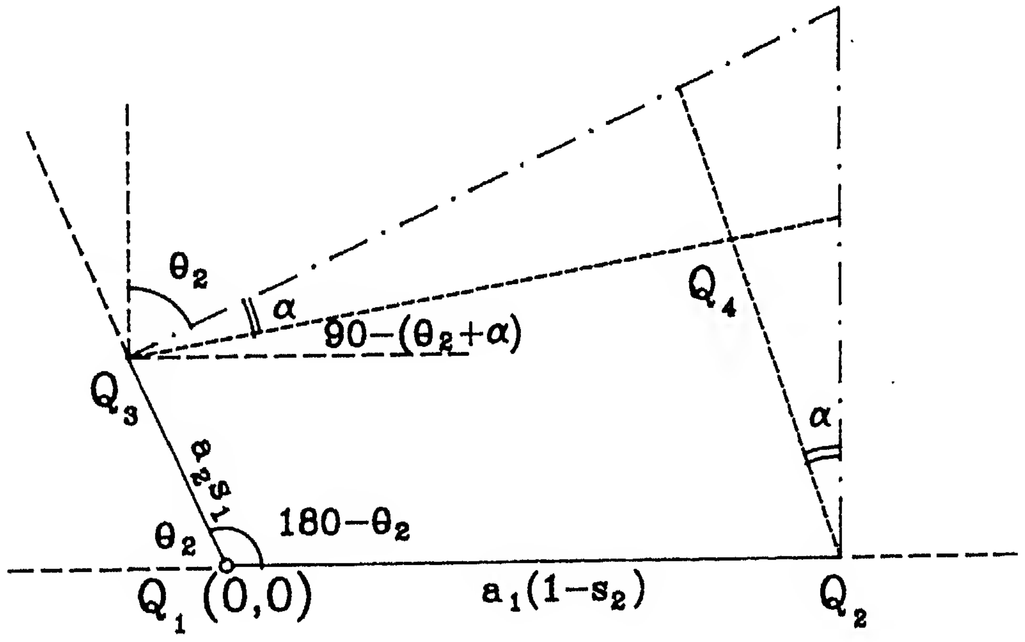


Figure 5.37: Finding the area of shadow area $\Delta_3$

where $q_1 = Q_1Q_2$, $q_2 = Q_2Q_3$, $q_3 = Q_3Q_4$, $q_4 = Q_4Q_1$ and $s = (q_1+q_2+q_3+q_4)/2$ is the semi-perimeter.

The area of $\Delta_4$ is found exactly as $\Delta_3$, but with $\theta_2$ replaced by $(-\theta_2)$ in all equations.

The objective function, $F(s_1, s_2)$, is formulated to give the best choice of $s_1$ and $s_2$, $0 \le s_1 \le s_2 \le 1.0$, for the joint *range* $0 \le \theta_2 \le \theta_{2,max}$ where $\theta_{2,max}$ is the limiting value of $\theta_2$. The choice of the objective function is adopted as

$$F(s_1, s_2) = \sum_{\theta_2=0}^{\theta_{2,max}} \Delta_{shadow}(\theta_2) \tag{5.53}$$

Table 5.3 lists the parameters $s_1$ and $s_2$ for which the objective function has been found to be minimum for $\theta_{2,max}$ ranging from 0° to 180° at 2° intervals and taking $a_1 = 20$ units, $a_2 = 20$ units, $\alpha = 30°$.

It is observed that the best position for placing sensors varies with the joint angle $\theta_{2,max}$. A placement that works well for all angles is therefore not possible.

A value is chosen that is best for the range of joint values that most frequently occur. It is seen that $s_1 = 0.2$ and $s_2 = 0.8$ (approximately) looks reasonable for the range 0° to 90°. The final choice of the sensor location on a link is shown in Figure 5.38.

A convention is now adopted for naming sensors. For each link, first imagine a vector originating from $h_i$ and pointing at $t_i$. All sensors falling to the left of this vector will be marked as left sensors, while those falling to the right as right sensors. The sensor on link $i$ falling on the left side located towards the head of the link will be labelled $C_{i,Lh}$ while the one falling on the right side and towards the tail of the link will be labelled as $C_{i,Rt}$. The four sensors on each link are, therefore $C_{i,Rh}$, $C_{i,Rt}$, $C_{i,Lh}$ and $C_{i,Lt}$ as shown in Figure 5.38.

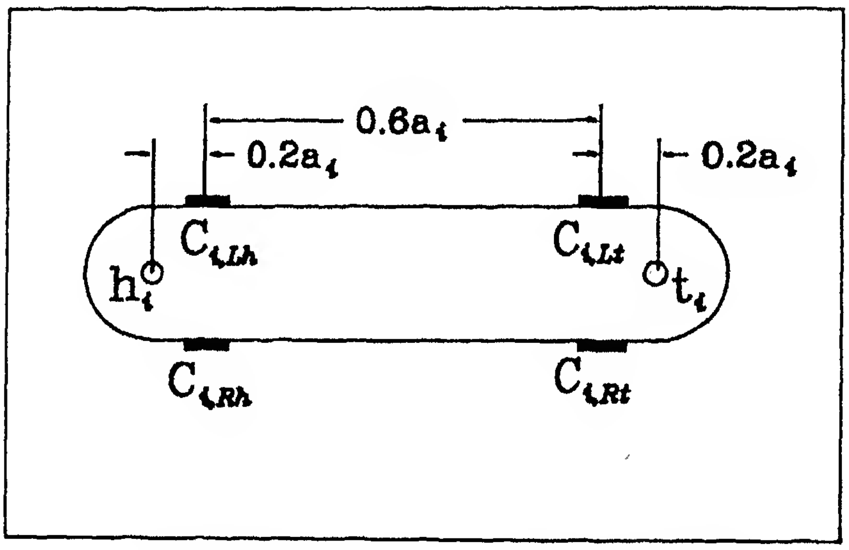


Figure 5.38: Final choice of sensor locations on a link

## 5.8 Suggestion for Building Perceived World Model

In the earlier sections the point obstacles were considered as inputs to the algorithm. This section suggests a method of combining sensory readings to build a perceived model of the world consisting of the point obstacles. This proposal should be taken only as a suggestion and must not be considered as binding. The subsequent analysis of the point obstacles through a fuzzy controller, however, remains unaffected by the changes in the mode of construction of the perceived world model.

The data provided by a single sensor is insufficient to estimate with certainty the position of an obstacle. It is thus wise to combine the data provided by multiple sensors to build a perceived "*image*" of the obstacle around the manipulator. This *image* is used later to estimate the direction of the obstacle with respect to each link so that fuzzy rules can be effectively applied. To distinguish the "Perceived World Model" from the actual world, the latter is called the "Physical World".

Figure 5.39 shows a four-link manipulator near an unidentified obstacle in the *Physical World.* All sensors are marked in this figure. It is expected that sensors

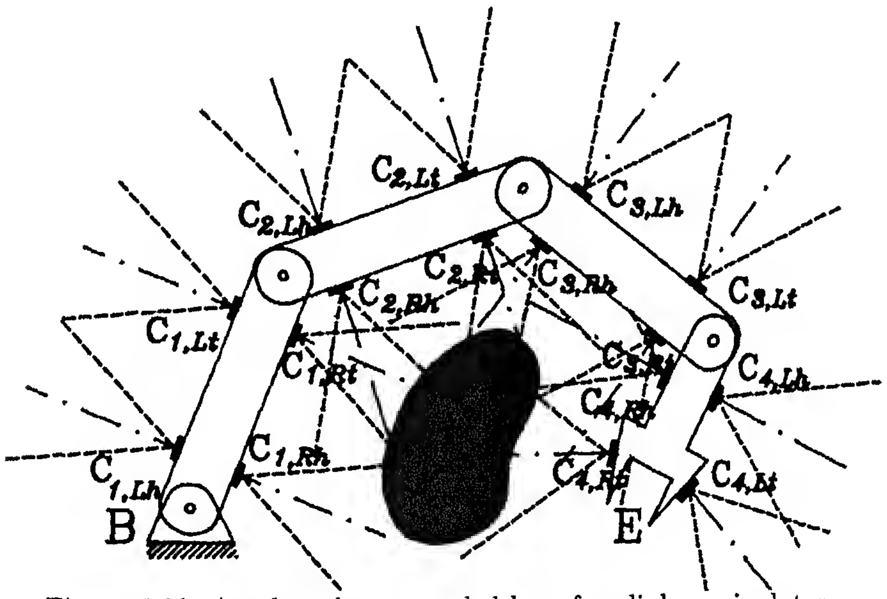


Figure 5.39: An obstacle surrounded by a four-link manipulator

$C_{1,Rh}$, $C_{1,Rt}$, $C_{2,Rh}$, $C_{2,Rt}$, $C_{3,Rh}$, $C_{3,Rt}$, $C_{4,Rh}$ and $C_{4,Rt}$ will detect the obstacle. These sensors as called the *activated sensors.* Since it is desired to build an image of this obstacle, a *pair* of sensors is chosen at a time, to build a piece wise image. The piece wise image will be formed by assimilating information received by pairs of activated sensors that are most likely seeing the *same* face of the obstacle. For example, in Figure 5.39, it would be inadvisable to build an image by pairing up sensors $C_{1,Rt}$ and $C_{3,Rt}$ since they are not seeing the same face of the obstacle. To be on the safe side, it is decided to choose only pairs of sensors that are mounted *adjacent* to each other on the link. In general, adjacent pairs are $C_{i,Rh}$ and $C_{i,Rt}$, $C_{i,Lh}$ and $C_{i,Lt}$, $C_{i,Rt}$ and $C_{i+1,Rh}$, and $C_{i,Lt}$ and $C_{i+1,Lh}$.

The links $L_1$ and $L_2$ are shown enlarged in Figure 5.40 with the adjacent active sensors. To form the image of the embedded obstacle, we consider the

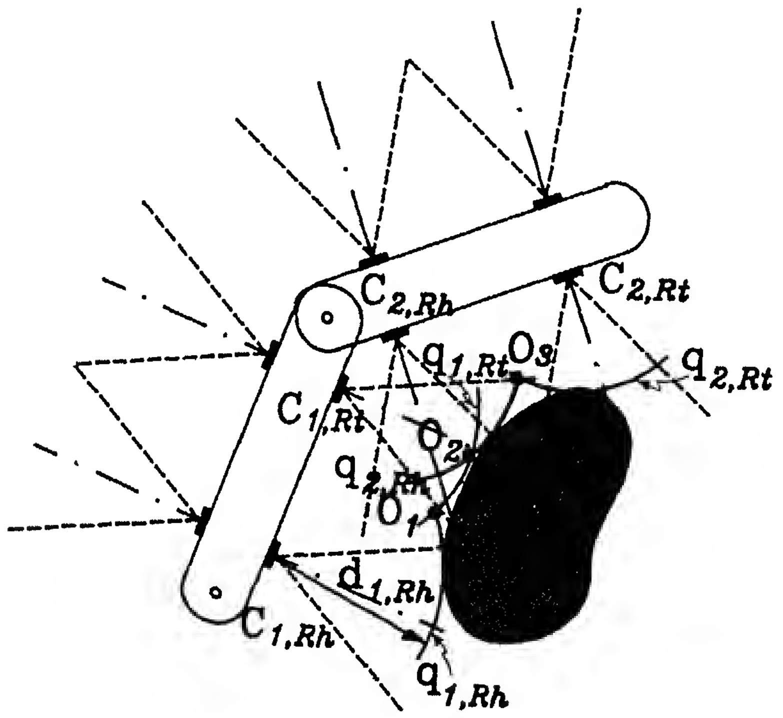


Figure 5.40: Forming the point obstacles from the physical obstacle

pairs $C_{1,Rh}$ and $C_{1,Rt}$, then $C_{1,Rt}$ and $C_{2,Rh}$ then finally $C_{1,Rh}$ and $C_{2,Rt}$. Since sensor $C_{1,Rh}$ detects the obstacle at a distance of $d_{1,Rh}$ an arc $q_{1,Rh}$ is drawn to represent the boundary of the obstacle as perceived by $C_{1,Rh}$ (see Figure 5.40). The intersection of $q_{1,Rh}$ with $q_{1,Rt}$ gives a point $O_1$ which is thought to be the common boundary point of the obstacle perceived by both sensors $C_{1,Rh}$ and $C_{1,Rt}$. Similarly, the intersection of arcs $q_{1,Rt}$ and $q_{2,Rh}$ gives the intersection point $O_2$. The intersection point $O_3$ is obtained by considering arcs $q_{2,Rh}$ and $q_{2,Rt}$. From the point of view of the collision avoidance algorithm, the points of intersection $O_j$ are as good as individual obstacles, and give sufficient information to estimate the exact orientation of the obstacle seen from a particular link. If the obstacle cannot be sensed by two sensors, as in Figure 5.41, the obstacle is assumed to be located exactly at the point of intersection of arc $q_{1,Rh}$ and the normal vector of the sensor $C_{1,Rh}$. In Figure 5.41 this point is shown as $O_1$.

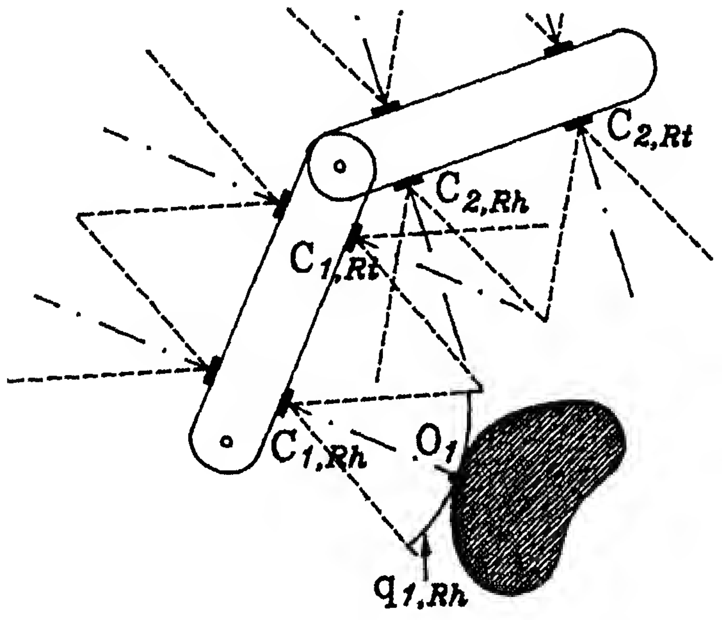


Figure 5.41: Image of an obstacle detected by a lone sensor

It is possible that in some strange configuration, one link of the manipulator will be detected as an obstacle from another link of the same manipulator. This point is interesting because it has often been ignored by purely geometric planners [51]. One such configuration is shown in Figure 5.42 where link 4 acts as an obstacle to link 1 and link 2. This fact cannot be ignored because it leads to valid collision cases. The links of the same manipulator are thus perceived as obstacles and a corrective action to *unfold* the manipulator under such circumstances is undertaken. An example which demonstrates this point is now given.

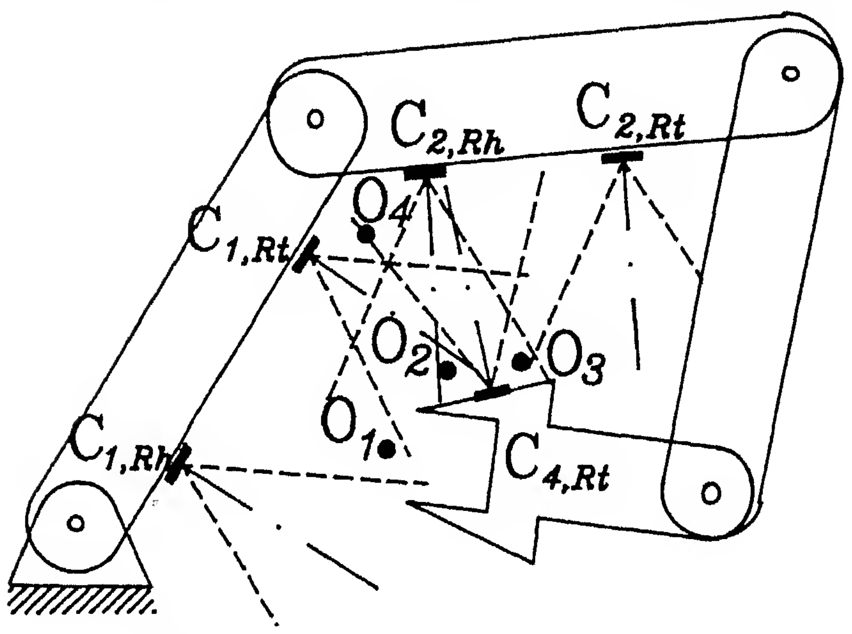


Figure 5.42: Link being perceived as an obstacle

### 5.8.1 Example with manipulator arms behaving as obstacles

Herein we consider an example which requires the manipulator to track a path that forces the links to get entangled among themselves if only the minimum norm solution is used. Figure 5.43 shows failure of achieving collision avoidance when the sensors are made inactive. The reorientation of links due to escape motion vector does not take place in this case. Figure 5.44 shows the same path being tracked by the manipulator but with sensors made active. It is observed that the path can be tracked without collision because of the timely reorientation of links when they come too close to each other.

It is clear from the preceding discussion that the point obstacles $O_j$ generated, are not *universal* obstacles. In other words an obstacle, say $O_j$, cannot be considered an obstacle by *all* links of the manipulator. An obstacle $O_j$ should be considered an obstacle only by links whose sensors have detected the obstacle, and have contributed to its formation. For example, in Figure 5.42 $O_1$ and $O_2$ are obstacles only for link 1 and link 2, but are not obstacles for link 4. Similarly obstacle $O_4$ is an obstacle for link 4, but not for link 2 or link 3. Thus, the obstacles in the

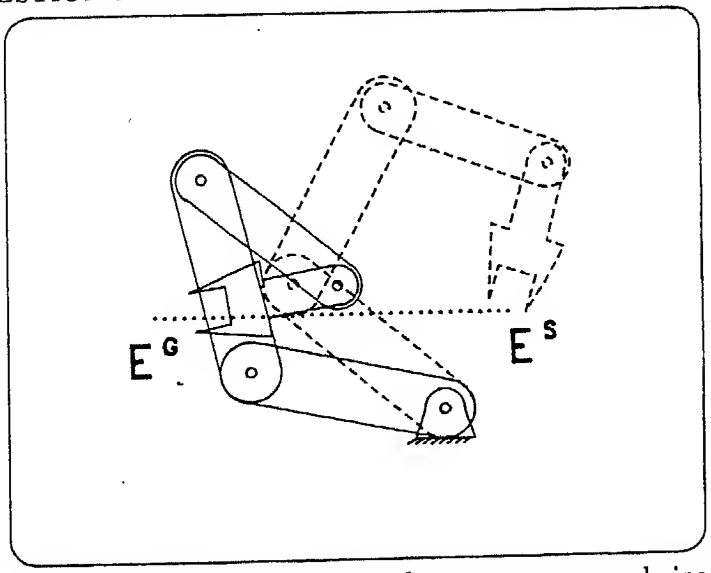


Figure 5.43: Entanglement of links when sensors are made inactive

*Perceived World Model* may or may not have any resemblance with the obstacles in the *Physical World.*

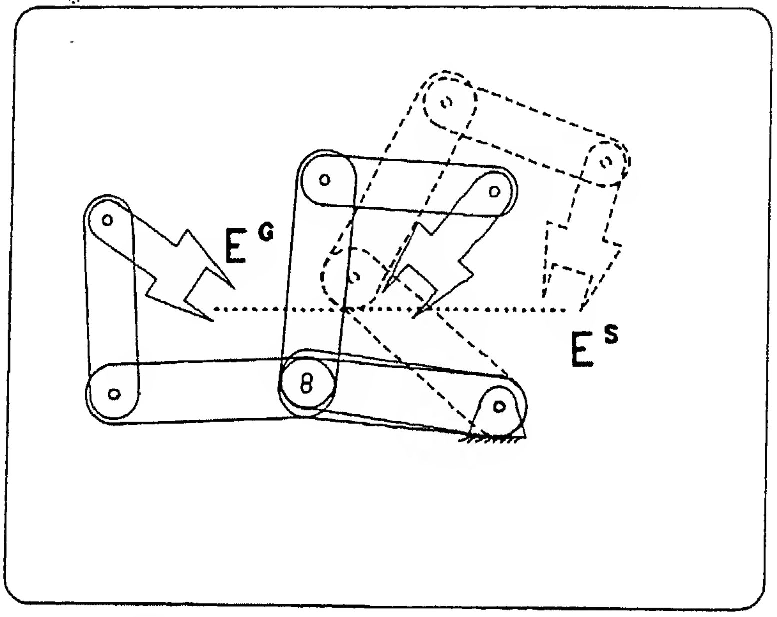


Figure 5.44: Solution of the four-link manipulator with sensors made active

| $\theta_{2,max}$ (degrees) | $s_1$ | $s_2$ | $\theta_{2,max}$ (degrees) | $s_1$ | $s_2$ |
|---|---|---|---|---|---|
| 0 | 0.14 | 0.86 | 92 | 0.19 | 0.82 |
| 4 | 0.14 | 0.86 | 96 | 0.22 | 0.85 |
| 8 | 0.14 | 0.86 | 100 | 0.22 | 0.84 |
| 12 | 0.14 | 0.86 | 104 | 0.17 | 0.79 |
| 16 | 0.13 | 0.86 | 108 | 0.18 | 0.79 |
| 20 | 0.13 | 0.86 | 112 | 0.19 | 0.8 |
| 24 | 0.13 | 0.86 | 116 | 0.19 | 0.8 |
| 28 | 0.13 | 0.87 | 120 | 0.2 | 0.8 |
| 32 | 0.14 | 0.86 | 124 | 0.2 | 0.8 |
| 36 | 0.15 | 0.86 | 128 | 0.2 | 0.8 |
| 40 | 0.15 | 0.85 | 132 | 0.2 | 0.8 |
| 44 | 0.15 | 0.85 | 136 | 0.21 | 0.79 |
| 48 | 0.15 | 0.84 | 140 | 0.21 | 0.79 |
| 52 | 0.16 | 0.84 | 144 | 0.21 | 0.79 |
| 56 | 0.16 | 0.84 | 148 | 0.21 | 0.79 |
| 60 | 0.16 | 0.83 | 152 | 0.21 | 0.79 |
| 64 | 0.17 | 0.83 | 156 | 0.21 | 0.79 |
| 68 | 0.17 | 0.83 | 160 | 0.21 | 0.79 |
| 72 | 0.17 | 0.82 | 164 | 0.22 | 0.78 |
| 76 | 0.17 | 0.82 | 168 | 0.22 | 0.78 |
| 80 | 0.18 | 0.82 | 172 | 0.22 | 0.78 |
| 84 | 0.18 | 0.82 | 176 | 0.22 | 0.78 |
| 88 | 0.19 | 0.82 | 178 | 0.22 | 0.78 |
| 92 | 0.19 | 0.82 | 180 | 0.22 | 0.78 |

Table 5.3: Parameters $s_1$ and $s_2$ for minimum shadow area

*No escape,*
*No such thing; to dream*
*of new dimensions,*
*Checking checkmate by painting*
*the king's robe*
*So that he slides like a queen.*
[The Castle]
– ROBERT GRAVES

# Chapter 6

# Motion Planning of a 3-D Manipulator in a Dynamic Environment

## 6.1 Introduction

Most real world motion planning problems occur in three dimensional space. The concepts developed for obstacle avoidance of manipulators in Chapter 5 may be extended to cover three dimensional manipulators. The dimensionality of the problem increases the complexity of some issues such as sensing the environment and modelling the robot. On the other hand, higher dimensionality offers more ways of escape for the manipulator than in the two-dimensional case.

This chapter deals with modelling the 3-D environment, the manipulator and its sensors for simulation, task execution in a dynamically changing unknown environment using sensors with fuzzy logic, and demonstration of a few examples using the PUMA-560 robot.

## 6.2 Problem Statement

The objective of the problem is similar to that in Chapter 5. A 3-D manipulator working in an unknown environment must execute a given task while avoiding collision with its neighbouring obstacles, if any. The *task* is specified in two forms. In "path-preference" problems the task is to trace a given trajectory while in "point-to-point" problems the task is to move from a given starting position to another goal location. The starting position of the end-effector is denoted as $E^S$ and the goal position as $E^G$. In *path-preference* problems the path is depicted by a smooth curve from $E^S$ to $E^G$. In *point-to-point* problems the sequence of intermediate goal points are numbered $E^{G1}$, $E^{G2}$, ..., $E^{Gp}$.

In this chapter, the solution strategy is demonstrated by simulation of an actual industrial robot — the PUMA-560 manipulator. All modelling in subsequent sections will therefore pertain to only the PUMA-560 robot.

The mathematical description of the PUMA-560 robot is given in Appendix A. This includes setting up the link coordinate frames, formulation of the transformation matrices, derivation of the Jacobian terms, solution of the inverse kinematic problem by a geometric method and solution of the inverse dynamic problem by the Newton-Euler method.

## 6.3 Fuzzy Rules for Collision Avoidance of Links

The links and joints of the PUMA-560 robot are shown in Figure 6.1. For convenience, in this chapter, the position of the base coordinate system is represented as $B$ (base of robot), and the position of the tip of the end-effector as F (for 'finger'). Some more points are defined on the manipulator to act as spots for positioning the escape vectors. These points are located near the *joints* and are named the 'wrist', W, 'elbow', E and 'shoulder', S (to conform with human arm geometry). The position of the end arm is called wrist, the ending point of link 2 is the elbow and the ending point of link 1 is the shoulder. The points B, S, E, W and F are called "*pivot points*" for convenience and are shown in Figure 6.1.

The identification of *pivot points* on the manipulator is done to locate the escape vectors. The reorientation of links is to be done on the basis of the escape

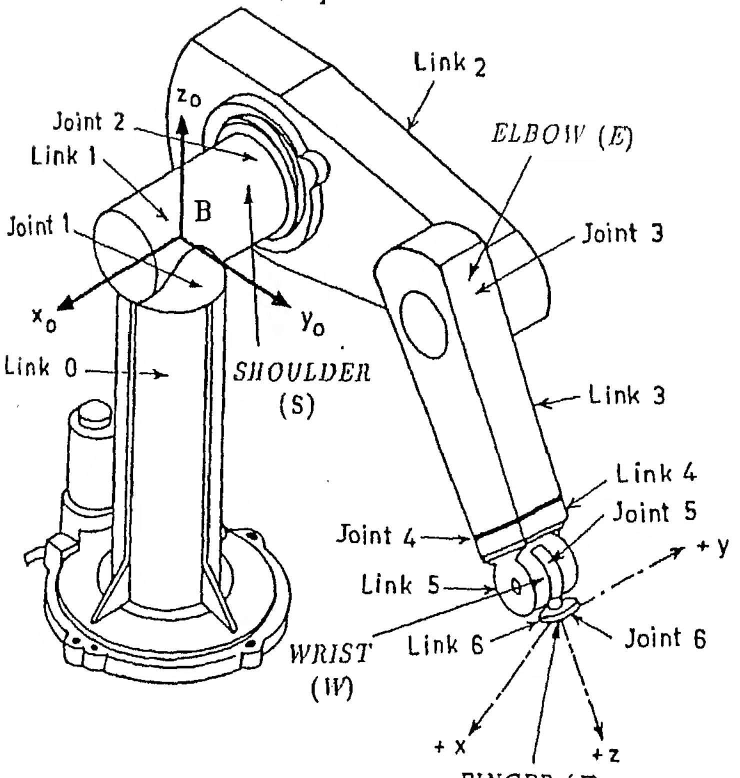


Figure 6.1: A PUMA robot illustrating joints and links

vectors acting at the pivot points. However, depending on the structure of the manipulator, all the escape vectors cannot be considered for reorienting the manipulator, because they might violate the kinematic constraints. The structure of the PUMA-560 allows a reconfiguration (keeping the end-effector static) only at the 'wrist' (because it is a wrist-partitioned manipulator). Thus for the purpose of this chapter, only the wrist, W and finger, F are considered as important pivot points. The other pivot points (S and E) are defined to demonstrate that such points may generally be defined for any industrial manipulator. The inconsequential ones may then be rejected depending on the geometry of the manipulator.

Collision avoidance of links is achieved by placing sensors at appropriate locations, then forming fuzzy rules to find *escape vectors* at selected positions on the robot's body, and finally interpreting the rules to move the links away from the nearby obstacles. Escape methods will not be easy to define for a three-dimensional manipulator, specially when dealing only with instantaneous data. One strategy is to move the manipulator links directly away from the sensed obstacle, but it does not guarantee in any way that joint limits will not be reached restricting any further movement. It is also possible that the direction of escape suggested by the sensors is impossible to achieve practically because this direction may not be attainable with the allowed directions of present joint movements. (since a revolute joint can only move about its axis of rotation). In such cases, the responsibility of moving the affected link must be passed on to other joints capable of moving in the suggested direction, or alternatively the direction of the escape vector itself can be changed to suit the joint characteristics.

In this section a sensor placement scheme is proposed, then a simple set of fuzzy rules are formulated resulting in escape vectors at the *wrist* and *finger*, and finally some methods are suggested for interpreting the escape vectors to bring about motion.

### 6.3.1 Placement of sensors

Guarding a three-dimensional volume requires a large number of sensors. Sensors are thus placed at the body of the manipulator scanning the volume that lies in its neighbourhood. Placing equal number of sensors at every link is not advisable because a link which is small in size needs to guard a smaller volume of space. Thus the number of sensors to be placed on a link is directly proportional to the size of the link. At the same time it is also to be noted that not all links are equally free to move (contrary to the two-dimensional problem presented in Chapter 5). Links which are close to the base of the robot has restricted movement. If sensors are installed on these links, it does not lead to any advantage because if an obstacle is detected, no action can be taken to *prevent* a collision, except perhaps freezing all movement. It is thus pointless to install too many sensors near the base of the robot.

Figure 6.2 shows a scheme of installing sensors on the body of the link and this scheme has been used in the simulation runs. Each sensor beam, which is directed outwards from the link, is specified by defining a normal vector. The exact positions of sensors for each link and the normal vectors of each sensor

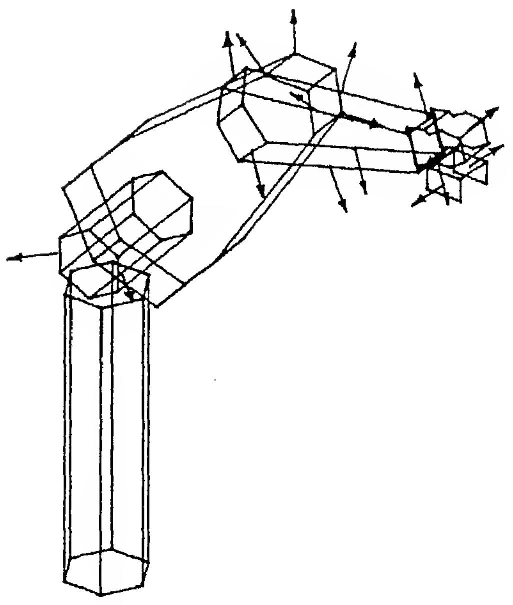


Figure 6.2: Proposed scheme for placement of sensors on PUMA

expressed in the coordinate system of the link, appears in Appendix F.

It may be noted here that with a different arrangement of sensors, it is straightforward to modify the algorithm to avoid collisions. In other words, the proposed scheme suffices to explain the underlying ideas of the fuzzy rule based algorithm.

## 6.3.2 Fuzzy rules for collision avoidance

The fuzzy rules that ascertain the escape vectors at the 'pivot points' of the manipulator are formulated in this section. Since the direction of the normal to the sensor decides the *direction* of escape, "*Direction*" is not included in the set of fuzzy variables any more. There is only one input parameter, namely "*Distance*" and the output parameter is also "*Distance*". The fuzzy rules can be simply put as shown in Table 6.1.

Viewed from a different perspective, the elimination of angle from the set of fuzzy rules (see Table 5.1 for an example which uses angle), effectively means that the *perceived world model* is not constructed any more, but it has been decided

| | IF | THEN |
|---|---|---|
| *Rule* | *Obstacle Distance is* | *Move link by Distance* |
| *1* | Very Near | Far |
| *2* | Near | Moderate distance |

Table 6.1: Set of rules for collision avoidance of links

to work in the *physical world.* This saves a lot of computational cost in finding the perceived world model from data received by large number of sensors in three dimensional space. Finding the perceived world model and then including angle has not been found to be fruitful enough to pay for the extra effort; because the manipulator itself is too restrictive to respond to the more *accurate* escape vectors resulting from the inclusion of the *perceived world model.*

## 6.4 Incremental Movement Schemes for PUMA-560

### 6.4.1 Movement in the presence of obstacles

The strategy to move the manipulator under the influence of obstacles remains same as that in the earlier chapters. The reading of a sensor is first fuzzified and fed into the fuzzy controller. The output quantity is defuzzified and results in a crisp value of distance that represents the intensity with which the point on the link (at which the sensor is located), must move away to avoid a collision. The direction of escape motion is made exactly opposite to the normal vector of the sensor. For the PUMA-560, the escape vectors of all sensors are transferred to two *pivot points* — the wrist W and the finger F. The *resultant* influence of all sensor readings at these pivot points decides the net escape vector acting there. The escape vectors at W and F are now used to geometrically reconfigure the links so that an impending collision may be avoided.

Let the position of the pivot points S, W and F be represented as S $\equiv (x_S, y_S, z_S)$, W $\equiv (x_W, y_W, z_W)$ and F $\equiv (x_F, y_F, z_F)$ respectively (Refer to Figures 6.1 and 6.2). Further, let the position of a sensor be denoted by $C \equiv (x_C, y_C, z_C)$ and the magnitude of the escape vector found after defuzzification be denoted by $|\vec{C}_{Esc}|$. If the direction of the normal to the sensor $C$ is $\vec{\mathbf{c}} = c_x\hat{\imath} + c_y\hat{\jmath} + c_z\hat{k}$ then the escape vector of sensor $C$ may be represented in vectorial form as

$$\vec{C}_{Esc} = -\,|\,C_{Esc}\,|\,\frac{c_x\hat{\imath} + c_y\hat{\jmath} + c_z\hat{k}}{\sqrt{c_x^2 + c_y^2 + c_z^2}} \tag{6.1}$$

Equivalent escape vectors at the wrist and finger are done according to the following rules:

Escape vectors of sensors at links 1, 2 and 3 are transferred to the wrist W.

Escape vectors of sensors at links 4 and 6 (5 is invisible) are apportioned to the wrist W and finger F.

The escape vector at a sensor $C$ on either link 1, 2 or 3, represented as $\vec{C}_{Esc}$, is transferred to the wrist in the following manner (See Figure 6.3(a)).

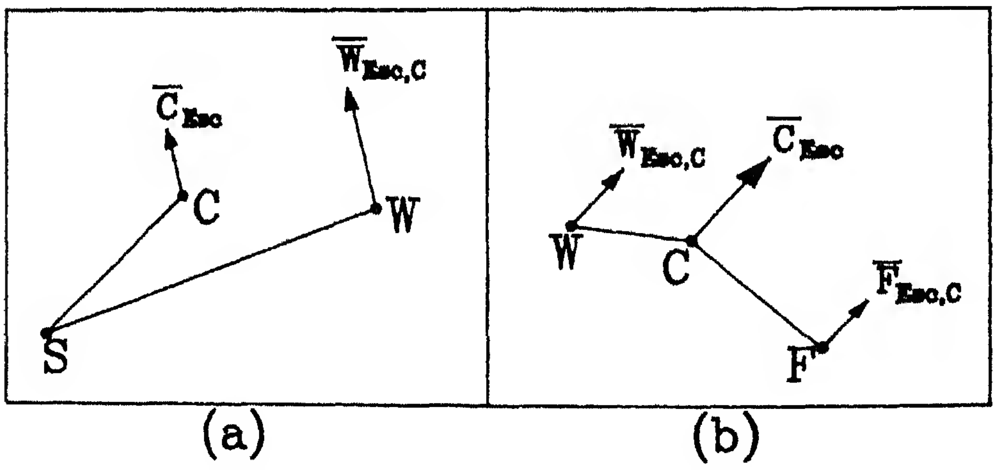


Figure 6.3: Finding equivalent escape vectors at the Wrist and Shoulder

The direction of the equivalent escape vector at W, called $\vec{W}_{Esc,C}$, is made same as the direction of $\vec{C}_{Esc}$. The magnitude of the escape vector is computed from

$$|\vec{W}_{Esc,C}| = |\vec{C}_{Esc}| \frac{\overline{WS}}{\overline{CS}} \tag{6.2}$$

The apportioning of an escape vector at link 4 or 6 is done on the basis of *magnitude* only, while the *direction* of the escape vector is retained. The magnitudes of the escape vectors $\vec{W}_{Esc}$ and $\vec{F}_{Esc}$ are obtained from the following equations (See Figure 6.3(b)).

$$|\vec{W}_{Esc,C}| = |\vec{C}_{Esc}| \frac{\overline{CF}}{\overline{CF}+\overline{CW}} \tag{6.3}$$

$$|\vec{F}_{Esc,C}| = |\vec{C}_{Esc}| \frac{\overline{CW}}{\overline{CF}+\overline{CW}} \tag{6.4}$$

The net escape vectors acting at the wrist and finger are the resultants of the escape vectors from all sensors. The net resultant of the escape vectors acting at these points is represented by $\vec{W}_{Esc}$ and $\vec{F}_{Esc}$ respectively.

### Geometric reorientation of links

A method to reorient the links of the manipulator based on the escape vectors is discussed here. The technique used for geometric reorientation is different for *path-preference* problems and *point-to-point* problems. In path-preference problems, the objective of the manipulator is to retain the end-effector on the path while adjusting the links to move away from the obstacle. In point-to-point problems however, there is no such restriction.

**Relocating the wrist for path-preference problems** Figure 6.4 shows the escape vector acting at W suggesting it to move to new position W′. A vector $\overline{FW'}$ is constructed and distance $d_6$ (remember $d_6 = \overline{FW}$ originally) is cut along $\overline{FW'}$ yielding the new position of the wrist W″. The geometric method of inverse kinematics discussed in Section A.3 is now used to find the joint locations of the PUMA.

**Relocating the wrist for point-to-point problems** In point-to-point problems the path of the end-effector is not fixed. According to our convention, the

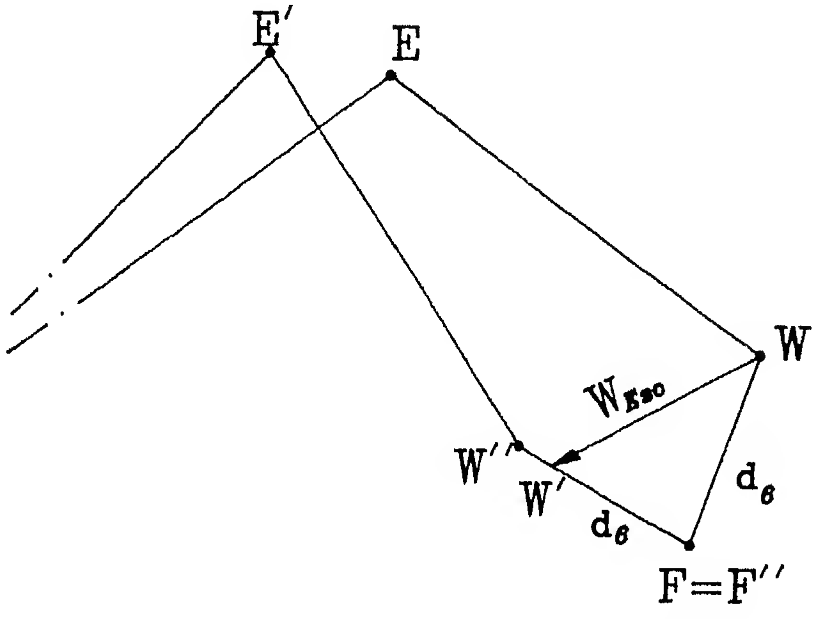


Figure 6.4: Relocating the wrist to avoid collision for path-preference problems

end-effector moves in incremental steps towards the next goal position. This attractive force towards the goal may be represented as a vector. At the same time the escape vector acting on the finger F restricts (or accelerates) the motion of the end-effector. The resultant of the two vectors decides the effective motion of the end-effector. Let the effective motion vector acting at the end-effector be represented by $\vec{F}_{Eff}$ as represented in Figure 6.5. The new suggested position of the finger is then F$'$. The escape motion vector acting at the wrist is $\vec{W}_{Esc}$ and the new suggested position of the wrist is W$'$. In point-to-point problems, the wrist is first positioned at W$'$. Then the vector $\overline{\mathrm{W'F'}}$ is found and a length $d_6$ is cut along the line $\overline{\mathrm{W'F'}}$ from W$'$ yielding the new position of the finger F$''$. The geometric method of Section A.3 is now used to solve the inverse kinematic problem for the PUMA.

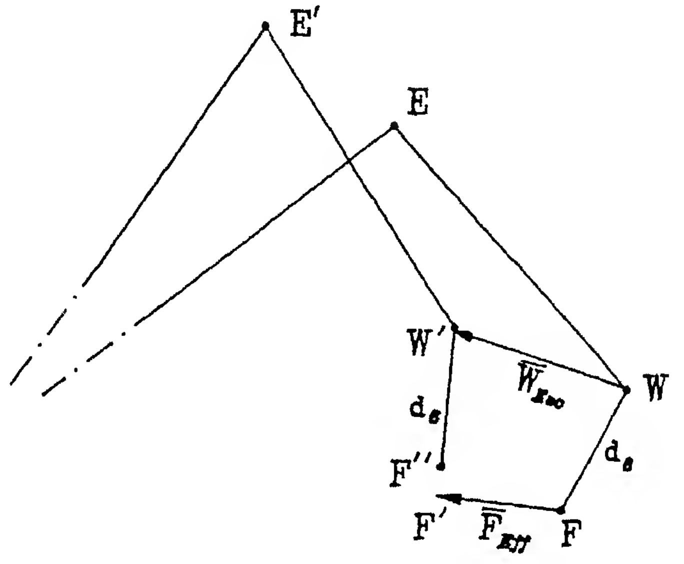


Figure 6.5: Relocating the wrist to avoid collision for point-to-point problems

**Strategy for extremely dangerous environments** Under extremely dangerous circumstances (signaled by large escape vectors of shoulder, elbow, wrist etc.), the manipulator must sacrifice its current goal and attempt to satisfy its primary objective of collision avoidance. In this situation, the attractive force towards the goal is ignored and only the motion under the influence of the escape vectors is enforced. The resulting motion is expected to make the manipulator *drift* away from the obstacle(s) thus reducing the danger of collision. The solution strategy of the point-to-point problem case is adopted to find the joint values.

### Checking motion limits of joints

The escape vectors at the wrist and finger are used to reconfigure the manipulator links geometrically. When the magnitude of the escape vectors become very large, it indicates high chance of collision and demands quick movement of the manipulator joints. However, the kinematic constraints like joint value and joint velocity, and the dynamic constraints like joint torque have to be kept within limits. The joint velocities and the joint torques are calculated at each step of execution and checked. The joint torques are evaluated on the basis of the procedure outlined in Section A.5 using the parameters of Table A.2. If the constraints on joints 1, 2 or 3 are violated, then the magnitude of the escape vector at the *wrist* W, is reduced by half and the arm (comprising links 1, 2 and 3) is reconfigured. If the constraints on joints 4, 5 or 6 are violated, then the magnitude of the escape vector at the *finger* F, is reduced by half and the wrist is reconfigured.

### 6.4.2 Movement in the absence of obstacles

The PUMA-560 is a six degree of freedom robot and thus has no redundancy for operations that need both positioning and orienting. However, when the nature of the task does not require orienting, the PUMA-560 behaves as a redundant manipulator. Under such circumstances, a solution has to be found by imposing external constraints. When the manipulator must avoid obstacles in the environment, the obstacle avoidance strategies itself provide the external constraints. When no external constraints are available then the inverse position solution is obtained by defining an appropriate approach vector and the inverse incremental solution is obtained by using the minimum norm solution [109]. In the following sections two solution strategies are investigated separately.

#### A. Assigning an approach vector for a path

When adequate constraints on the manipulator configuration are absent, the complete solution is obtained by providing extra constraints depending on the geometry of the job. For example, when the end-effector is tracing a path, an extra constraint could be imposed that the approach vector, $\vec{a}$ of the end-effector be normal to the trajectory. Additionally, it might also be imposed that the sliding vector, $\vec{s}$ must be along the tangent to the path. A proposal for easily determining the hand vectors $[\vec{n}, \vec{s}, \vec{a}]$ without considering the geometry of the manipulator is given in Section 6.6.2. After determining the hand vectors, the geometric solution of Section A.3 is used to solve for the manipulator.

### B. Minimum Norm solution for PUMA

In the absence of constraints requiring orientation of the end-effector, the PUMA-560 behaves as a redundant robot. Thus the task space vector in this case is $\mathbf{x} = [x\ y\ z]^T$ which denotes the position vector of the end-effector with respect to the base coordinate frame $(\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$. The position vector in joint space is represented as $\mathbf{q} = [\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6]^T$ as before.

The differential relationship between the task space vector and the joint vector is given by

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \end{bmatrix} \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta\theta_3 \\ \Delta\theta_4 \\ \Delta\theta_5 \\ \Delta\theta_6 \end{bmatrix} \tag{6.5}$$

where the elements of the Jacobian matrix are given by (A.54).

The Lagrangian $L$, which serves as the optimization function is now formed as

$$\begin{aligned} L \ = \ & (\Delta\theta_1)^2 + (\Delta\theta_2)^2 + (\Delta\theta_3)^2 + (\Delta\theta_4)^2 + (\Delta\theta_5)^2 + (\Delta\theta_6)^2 \\ & +\lambda_1(J_{11}\Delta\theta_1 + J_{12}\Delta\theta_2 + J_{13}\Delta\theta_3 + J_{14}\Delta\theta_4 + J_{15}\Delta\theta_5 + J_{16}\Delta\theta_6 - \Delta x) \\ & +\lambda_2(J_{21}\Delta\theta_1 + J_{22}\Delta\theta_2 + J_{23}\Delta\theta_3 + J_{24}\Delta\theta_4 + J_{25}\Delta\theta_5 + J_{26}\Delta\theta_6 - \Delta y) \\ & +\lambda_3(J_{31}\Delta\theta_1 + J_{32}\Delta\theta_2 + J_{33}\Delta\theta_3 + J_{34}\Delta\theta_4 + J_{35}\Delta\theta_5 + J_{36}\Delta\theta_6 - \Delta z) \end{aligned} \tag{6.6}$$

To minimize $L$, the following nine conditions must be satisfied,

$$\begin{aligned} \frac{\partial L}{\partial \lambda_1} &= 0 \\ \frac{\partial L}{\partial \lambda_2} &= 0 \\ \frac{\partial L}{\partial \lambda_3} &= 0 \\ \frac{\partial L}{\partial (\Delta\theta_1)} &= 0 \end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial(\Delta\theta_2)} &= 0 \\ \frac{\partial L}{\partial(\Delta\theta_3)} &= 0 \\ \frac{\partial L}{\partial(\Delta\theta_4)} &= 0 \\ \frac{\partial L}{\partial(\Delta\theta_5)} &= 0 \\ \frac{\partial L}{\partial(\Delta\theta_6)} &= 0\end{aligned} \tag{6.7}$$

which gives rise to the following nine equations:

$$\begin{aligned}J_{11}\Delta\theta_1 + J_{12}\Delta\theta_2 + J_{13}\Delta\theta_3 + J_{14}\Delta\theta_4 + J_{15}\Delta\theta_5 + J_{16}\Delta\theta_6 - \Delta x &= 0 \\ J_{21}\Delta\theta_1 + J_{22}\Delta\theta_2 + J_{23}\Delta\theta_3 + J_{24}\Delta\theta_4 + J_{25}\Delta\theta_5 + J_{26}\Delta\theta_6 - \Delta y &= 0 \\ J_{31}\Delta\theta_1 + J_{32}\Delta\theta_2 + J_{33}\Delta\theta_3 + J_{34}\Delta\theta_4 + J_{35}\Delta\theta_5 + J_{36}\Delta\theta_6 - \Delta z &= 0 \\ 2\Delta\theta_1 + \lambda_1 J_{11} + \lambda_2 J_{21} + \lambda_3 J_{31} &= 0 \\ 2\Delta\theta_2 + \lambda_1 J_{12} + \lambda_2 J_{22} + \lambda_3 J_{32} &= 0 \\ 2\Delta\theta_3 + \lambda_1 J_{13} + \lambda_2 J_{23} + \lambda_3 J_{33} &= 0 \\ 2\Delta\theta_4 + \lambda_1 J_{14} + \lambda_2 J_{24} + \lambda_3 J_{34} &= 0 \\ 2\Delta\theta_5 + \lambda_1 J_{15} + \lambda_2 J_{25} + \lambda_3 J_{35} &= 0 \\ 2\Delta\theta_6 + \lambda_1 J_{16} + \lambda_2 J_{26} + \lambda_3 J_{36} &= 0\end{aligned} \tag{6.8}$$

In matrix form the above equations can be represented as

$$\begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} & 0 & 0 & 0 \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} & 0 & 0 & 0 \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & J_{11} & J_{21} & J_{31} \\ 0 & 2 & 0 & 0 & 0 & 0 & J_{12} & J_{22} & J_{32} \\ 0 & 0 & 2 & 0 & 0 & 0 & J_{13} & J_{23} & J_{33} \\ 0 & 0 & 0 & 2 & 0 & 0 & J_{14} & J_{24} & J_{34} \\ 0 & 0 & 0 & 0 & 2 & 0 & J_{15} & J_{25} & J_{35} \\ 0 & 0 & 0 & 0 & 0 & 2 & J_{16} & J_{26} & J_{36} \end{bmatrix} \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta\theta_3 \\ \Delta\theta_4 \\ \Delta\theta_5 \\ \Delta\theta_6 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (6.9)$$

Solution of (6.9) yields the incremental values of the joint angles, $\Delta\theta_1$, $\Delta\theta_2$, $\Delta\theta_3$, $\Delta\theta_4$, $\Delta\theta_5$ and $\Delta\theta_6$. The new values of the joint angles are obtained by adding $\Delta\theta_i$ to each $\theta_i$ $(i = 1, \ldots, 6)$.

## 6.5 Task Planning Scheme

This section considers two kinds of problems — a path-preference problem and a point-to-point problem, and proposes solution strategies for both while avoiding collision. The simulation results presented in this section were implemented on an IBM-AT running in the DOS environment. The execution time of the programs are for the IBM-AT running at 10 MHz.

In the first example a path-tracking problem is considered with a moving obstacle in the environment and the application of fuzzy rules with geometric reorientation is demonstrated through a simple problem. As described in Chapter 5, the motion of the manipulator under the influence of obstacles may be divided into two components, namely first moving the tip of the end-effector (i.e. the 'finger') using the minimum norm solution and second, reorienting the links of the manipulator (while keeping the end-effector static) to avoid a collision. The minimum norm solution is described in Section 6.4.2.

### 6.5.1 Example of path-preference problem

*Example 1 : Path-tracking problem with one moving obstacle.*

Figure 6.6 shows a PUMA-560 robot with a long stylus attached to the end-effector. The position shown is at the beginning of the path. The intended path

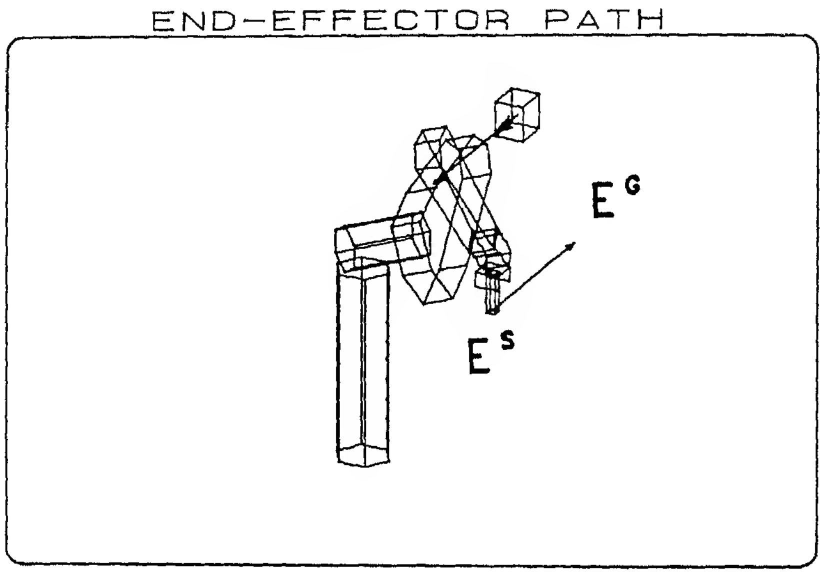


Figure 6.6: Manipulator at the beginning of path

of the end-effector is shown by the arrow line. The position of the obstacle is shown by the cuboid towards the top of the manipulator. The manipulator traces its path according to the minimum norm solution discussed in Section 6.4.2. To show a case of collision, the simulation was first done by deactivating the sensors. Figure 6.7 shows four views of the same scene when both the manipulator and the obstacle have advanced midway towards their goal position. It is clear from the figure that a collision has occurred between link 2 and the obstacle in the position shown. This failure has occurred because the sensors were deactivated and the collision avoidance algorithm was inactive. To show the effect of introducing the collision avoidance algorithm, the same problem is now solved with sensors made active. The position of the manipulator at the same instant is depicted in

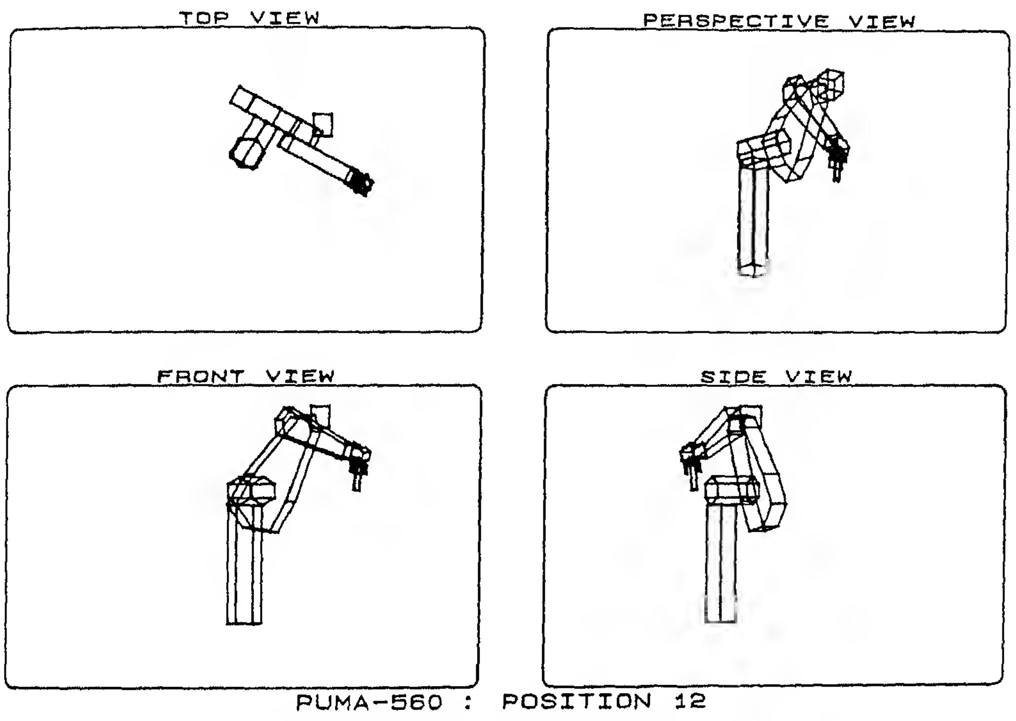


Figure 6.7: Position of the manipulator showing collision

Figure 6.8. In the view marked "SIDE VIEW" the obstacle is found to be clear of the manipulator links, indicating that a collision has been avoided[1]. This has been made possible because the escape vector at the wrist makes the manipulator to "dip" down so that it does not obstruct the path of the obstacle. In Figure 6.9 the manipulator is shown when it is nearest to the obstacle. The front and side views reveal that a collision has been successfully avoided. Finally Figure 6.10 shows the position attained by the manipulator at the end of the path. It is observed that the manipulator ends up reaching the final position with the stylus pointing up. This is in contrast to its beginning position where the stylus was pointing down — a change which has been brought about by the presence of the obstacle in its environment.

[1]It may be noted that a collision free configuration between two objects can be proved if *one* view can be shown where the two objects are completely separate.

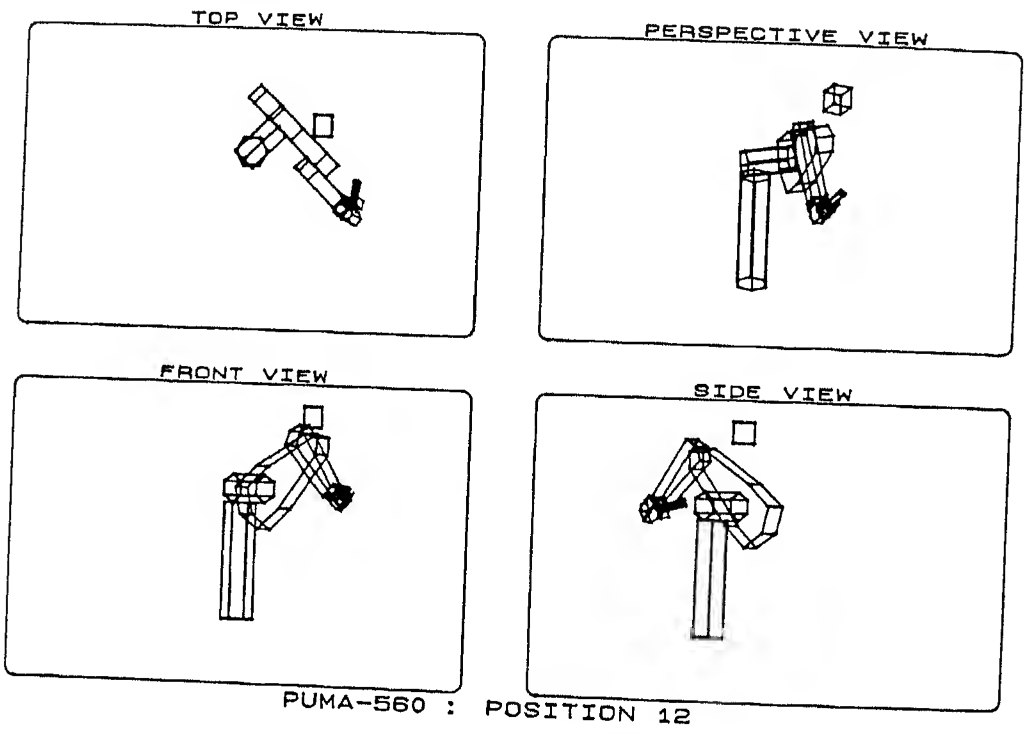


Figure 6.8: Same position of the manipulator showing collision-free configuration

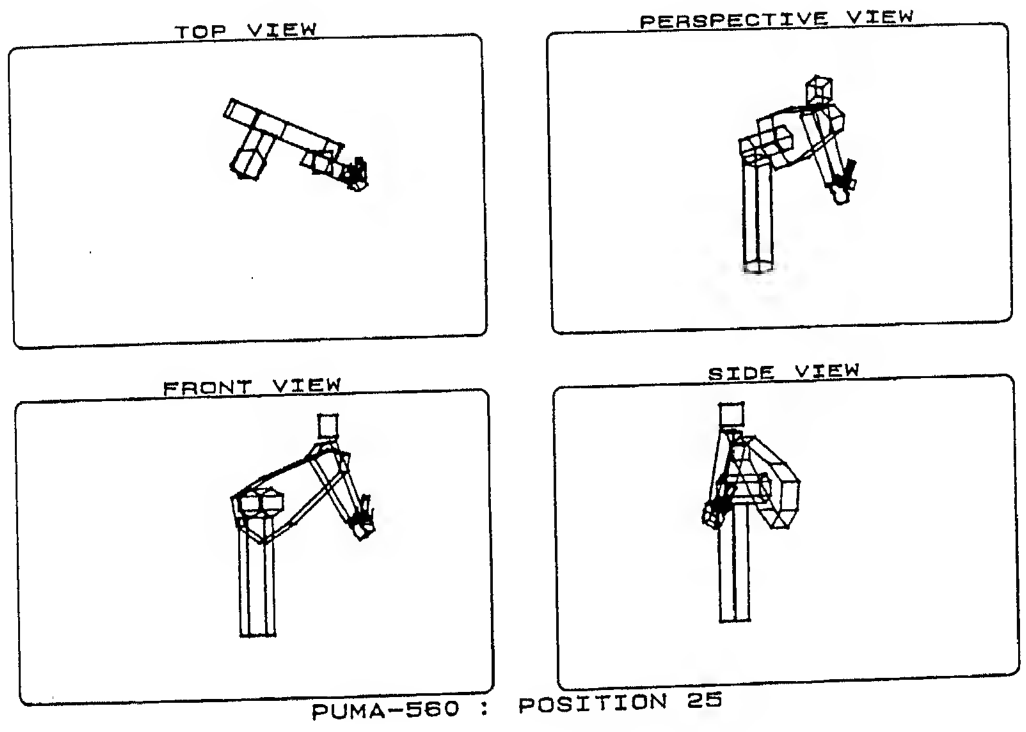


Figure 6.9: Position of the obstacle when it is nearest to the manipulator

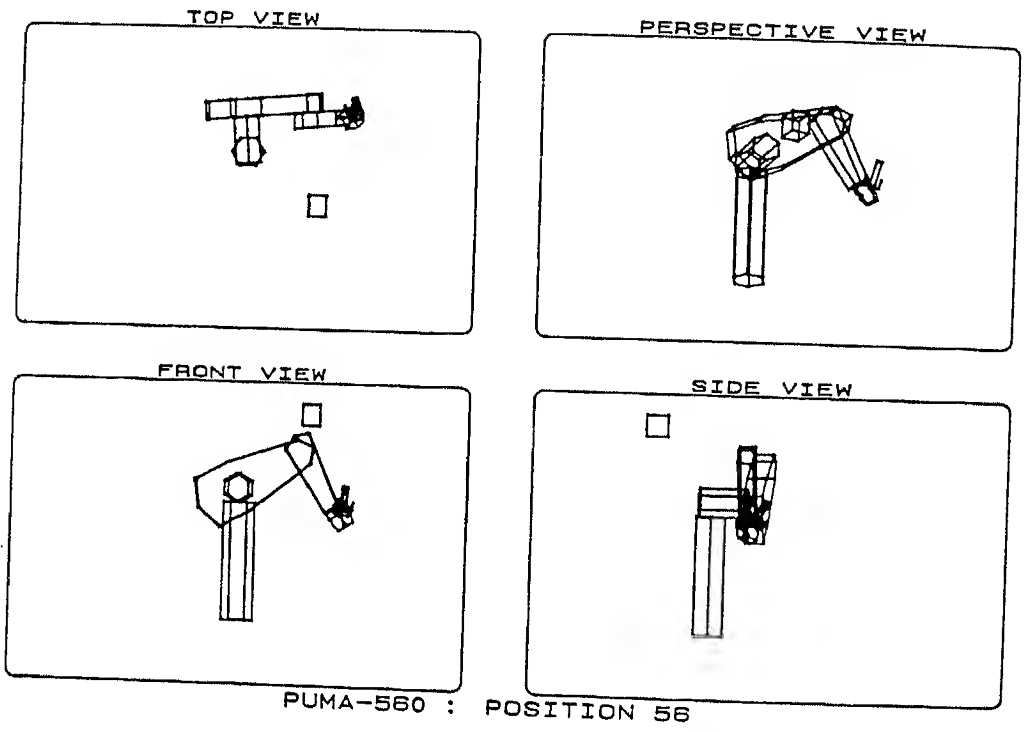


Figure 6.10: Position attained by the manipulator at the end of the operation

### 6.5.2 Examples of pick-and-place problems utilizing *state* change strategies

To explain the solution strategy for pick-and-place operations first a few *states* of the manipulator are defined, as was done in Chapter 5. To understand the *states* required for the pick-and-place operation, a more detailed analysis of the act of grasping is necessary.

#### The grasping sequence

The grasping sequence is explained with the help of Figure 6.11. Let $G_O$ be the

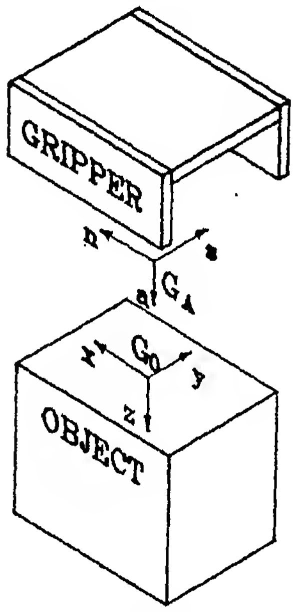


Figure 6.11: The grasping sequence

origin of the grasp object and let $G_A$ be defined to be the approach position of the grasp object as used in VAL [131]. The position of $G_A$ is generally defined by assigning a displacement along the negative $z$ direction of the coordinate system at $G_O$. In the following paragraphs a few states of the manipulator are introduced. These are the states *ORIENTING*, *APPROACHING* and *RETRACTING*. Two gripper states, *EMPTY* and *HOLDING*, are also introduced.

A grasping sequence (pick operation) comprises of the following steps:

- Bring the end-effector to coincide with point $G_A$. Make sure the gripper is open. The gripper state is now *EMPTY*.
- Orient the end-effector so that the hand vector $\vec{s}$ aligns with the $y$-axis of the object and the vector $\vec{a}$ aligns with the $z$-axis of the object. This operation describes the *ORIENTING* state.
- Move the end-effector slowly towards the point $G_O$ keeping the orientation fixed till the end-effector point coincides with the *pick* position $G_O$. This operation describes the *APPROACHING* state.
- Close the gripper. The gripper is now in *HOLDING* state.

The ungrasping (place operation) sequence comprises of the following steps:

- Move the end-effector to the desired approach point $G_A$.
- Orient the end-effector so that the hand vector $\vec{s}$ aligns with the $y$-axis of the desired placing position of the object and the vector $\vec{a}$ aligns with the $z$-axis. The manipulator is in *ORIENTING* state during this operation.
- Move the end-effector slowly towards the point $G_O$ keeping the orientation fixed till the end-effector point coincides with the desired *place* position $G_O$. The manipulator is in *APPROACHING* state during this operation.
- Release the gripper. The gripper is now in *EMPTY state*.
- Retract the gripper slowly keeping the orientation fixed so that the end-effector point again coincides with the approach point $G_A$. This operation describes the *RETRACTING* state.

It is obvious that the states *APPROACHING*, *RETRACTING* and *ORIENTING* are necessary to interact with an external body like the grasp object. However, since we are now dealing with a body that actually *touches* the robot (and later coalesces with it), no other external body must influence the motion of the manipulator when the handling operation is being done. Thus the manipulator is made "blind" to the motion of external bodies by deactivating the sensors used for obstacle avoidance during the states *APPROACHING*, *RETRACTING* and *ORIENTING*. In real situations, some force or proximity sensors should be employed for fine motion planning.

The other manipulator states required for this problem are enumerated as follows. The manipulator is in *NATURAL* state when it is not under the influence of any obstacle and is continuing normally towards the goal position. The manipulator is in *PAUSE* state when obstacles are first sensed in the environment. In this state the manipulator links are adjusted without change in end-effector position. The manipulator is in *DETOUR* state when it is advancing towards its goal as well as avoiding collision with neighbouring obstacles. When obstacles are sensed to be dangerously near the manipulator, it is in a state of *SURRENDER* in which the motion of the manipulator is decided solely by the motion of the surrounding obstacles. The present task of the manipulator is completely sacrificed to meet the prime requirement of collision avoidance. The manipulator is made to "drift" along with the obstacle and is expected to lead to a safer position from where *DETOUR*ing may be easier. It is to be noted that the meaning assigned to the *states* in the three dimensional problem is slightly different from their corresponding meaning in the two-dimensional case.

The algorithm for a state change operation is now mentioned in `procedure FindManipulatorState`.

```
procedure FindManipulatorState;
  begin
    if (Manipulator State = DEADLOCK) then
       Manipulator State := NATURAL;
    if ((Gripper State = EMPTY) and (Manipulator State = NATURAL) then
       if (end-effector has reached approach point of pick position) then
          Manipulator State := ORIENTING;
    if ((Gripper State = HOLDING) and (Manipulator State = NATURAL) then
       if (end-effector has reached approach point of place position) then
          Manipulator State := ORIENTING;
    if ((Gripper State = EMPTY) and (Manipulator State = ORIENTING)) then
       if (hand vector $\vec{s}$ is aligned with the initial Y-axis of object) then
          Manipulator State := APPROACHING;
    if ((Gripper State = HOLDING) and (Manipulator State = ORIENTING)) then
        if (hand vector $\vec{s}$ is aligned with the Y-axis of object's
                                final position) then
            Manipulator State := APPROACHING;
    if ((Gripper State = EMPTY) and (Manipulator State = APPROACHING)) then
        if (end-effector has reached picking position) then
            begin
                Gripper State := HOLDING;
                Manipulator State := NATURAL;
            end;
    if ((Manipulator State = NATURAL) and (danger level > NATURAL limit)) then
        Manipulator State := PAUSE;
    if ((Manipulator State = PAUSE) and (danger level > PAUSE limit)) then
        Manipulator State := DETOUR;
    if ((Manipulator State = DETOUR) and (danger level > DETOUR limit)) then
        Manipulator State := SURRENDER;
    if ((Manipulator State = SURRENDER) and (danger level < DETOUR limit)) then
        Manipulator State := DETOUR;
    if ((Manipulator State = DETOUR) and (danger level < NATURAL limit)) then
        Manipulator State := NATURAL;
    if ((Gripper State = HOLDING) and (Manipulator State = APPROACHING)) then
        if (end-effector has reached placing position) then
            begin
                Gripper State := EMPTY;
                Manipulator State := RETRACTING;
            end;
    if ((Gripper State = EMPTY) and (Manipulator State = RETRACTING)) then
        if (the end-effector has reached final approach position) then
            Manipulator State := NATURAL;
    if (Manipulator State = DETOUR) or (Manipulator State = PAUSE) then
        if (escape vectors have not changed for a long time) then
            Manipulator State := DEADLOCK;
end;
```

**Attachment of the grasp object to the end-effector**

In the *HOLDING* state of the gripper, the grasp object becomes a part of the robot. Hence, the grasp object acts as an extension of the end-effector. Table 6.2 describes the link parameters of the grasp object. The position of the gripper can

| Gripper's link coordinate parameters | | | | |
|---|---|---|---|---|
| *Joint i* | $\theta_i$ | $\alpha_i$ | $a_i$ | $d_i$ |
| ... | ... | ... | ... | ... |
| 6 | $\theta_6$ | 0 | 0 | 56.25 mm |
| 7 | 0 | 0 | 0 | *object length* |

Table 6.2: Link parameters for the gripper

be established by setting up a transformation matrix relating the end-effector to the gripper using parameters from Table 6.2.

*Example 2 : A pick-and-place operation.*
First an example is considered where a pick-and-place operation is demonstrated without any obstacle in the environment. Figure 6.12 shows a scene in which a block kept on a table A is to be picked up and placed on another table B. The figure shows the position of the manipulator when the end-effector is about to pick up the object. Figure 6.13 shows the position of the manipulator just after picking up the object. In Figure 6.14 the manipulator is midway on its way to the desired placing position. The position of the manipulator just before placing the block is shown in Figure 6.15 while its position just after placing is shown in Figure 6.16. The parking position of the manipulator after completion of the job is shown in Figure 6.17. This figure also shows the entire path traced by the end-effector with the arrow lines. It is observed that the path consists of straight line segments because the sensors do not indicate the presence of any obstacle in the environment. The only bodies which pose a risk of collision are the tables (on which the block rests) which are away from the manipulator's workspace.

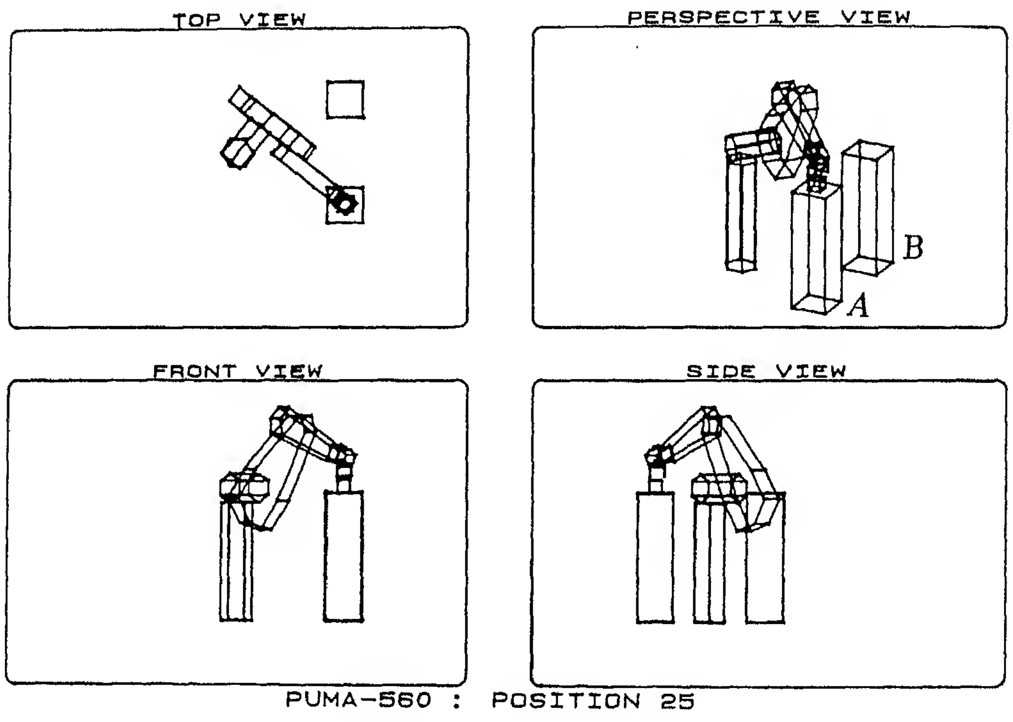


Figure 6.12: A pick-and-place operation : Manipulator just before picking

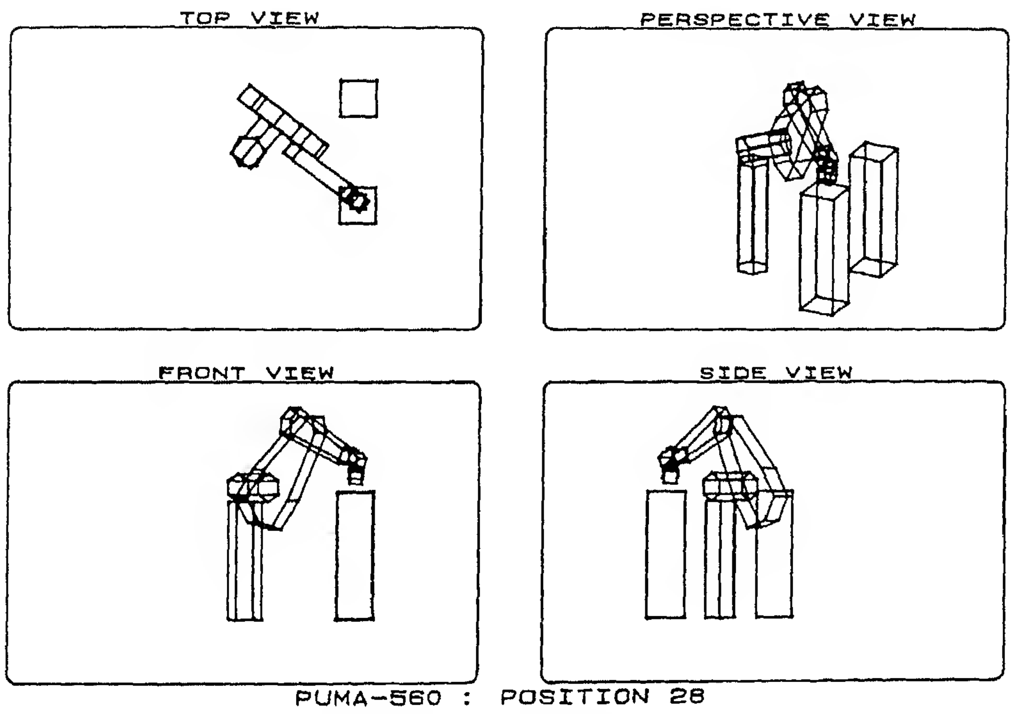


Figure 6.13: Position of manipulator just after picking up the block

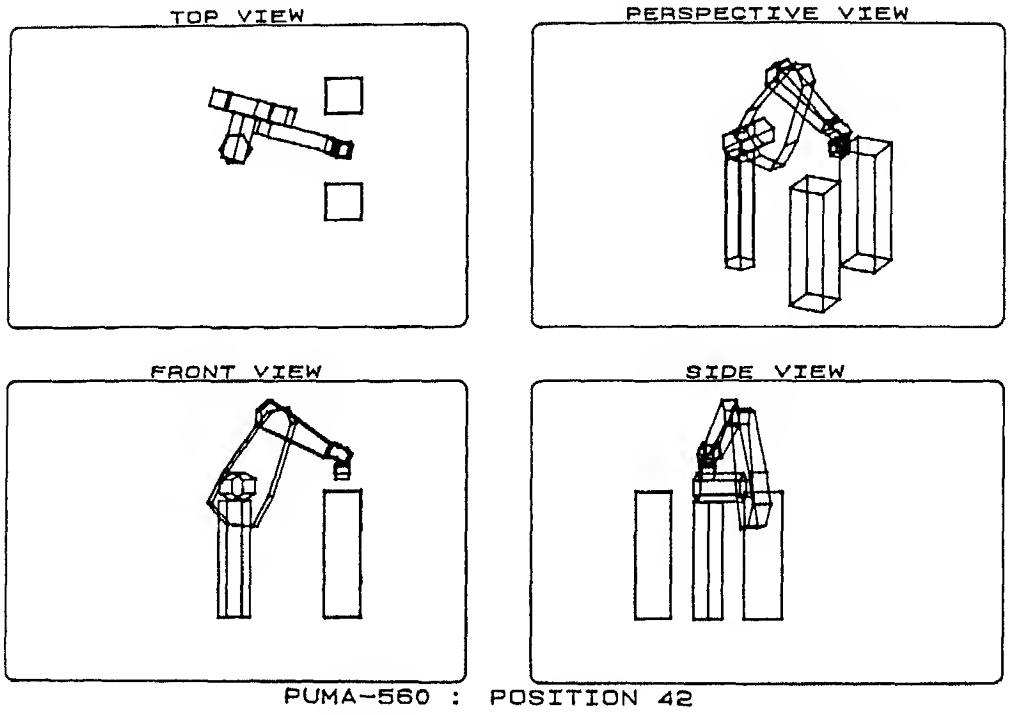


Figure 6.14: Manipulator's position midway between the pick and place locations

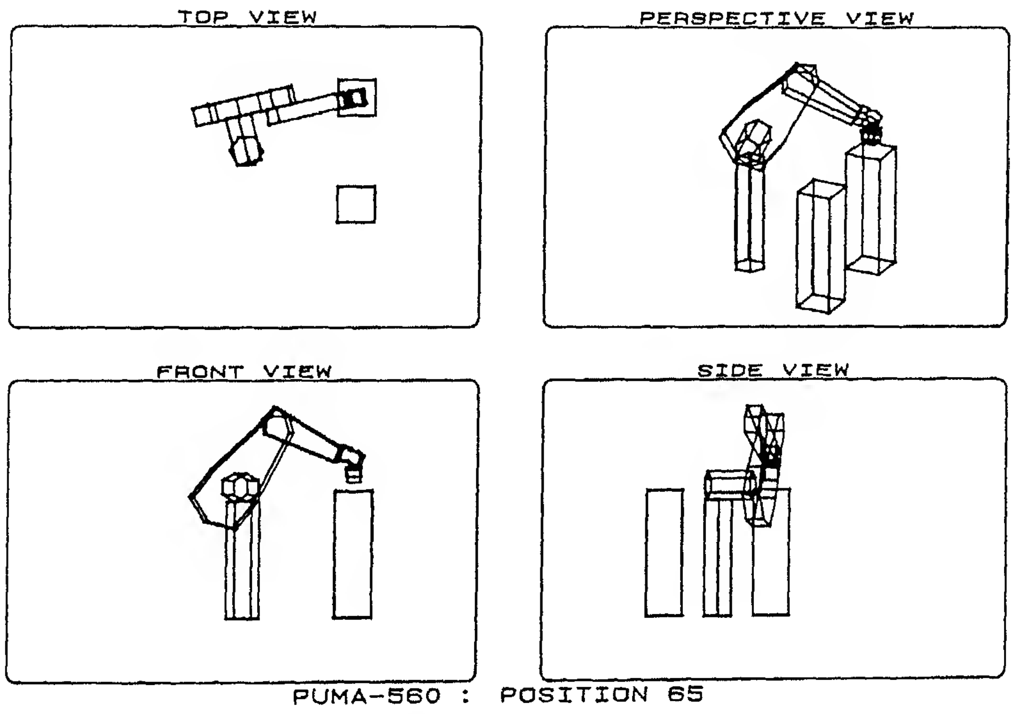


Figure 6.15: Position of the manipulator just before placing

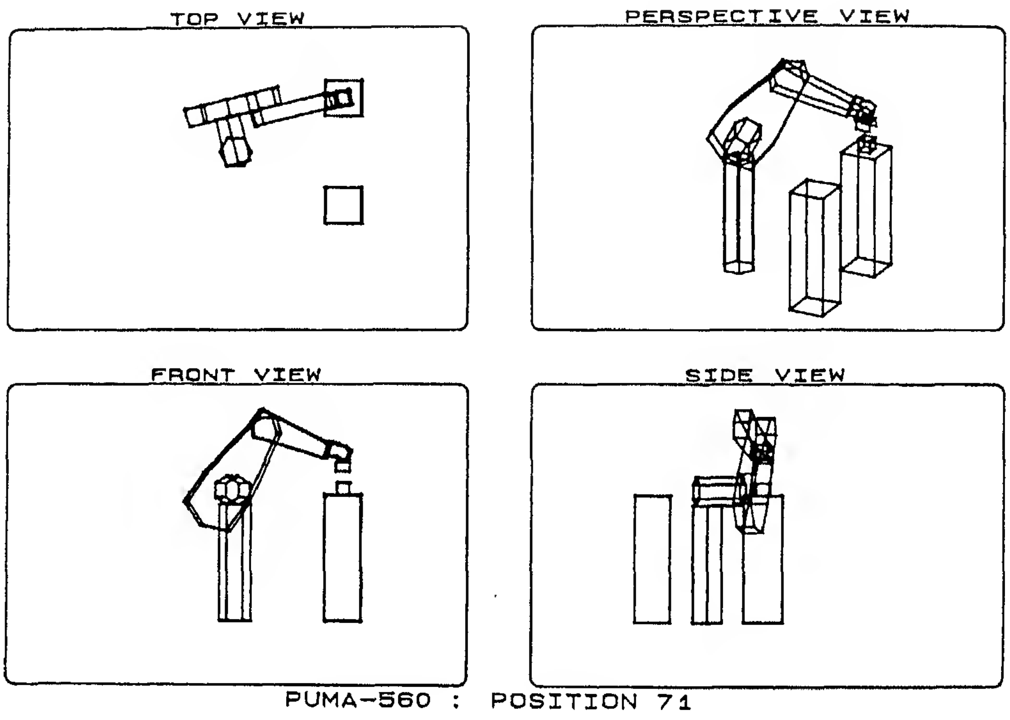


Figure 6.16: Position of the manipulator just after placing

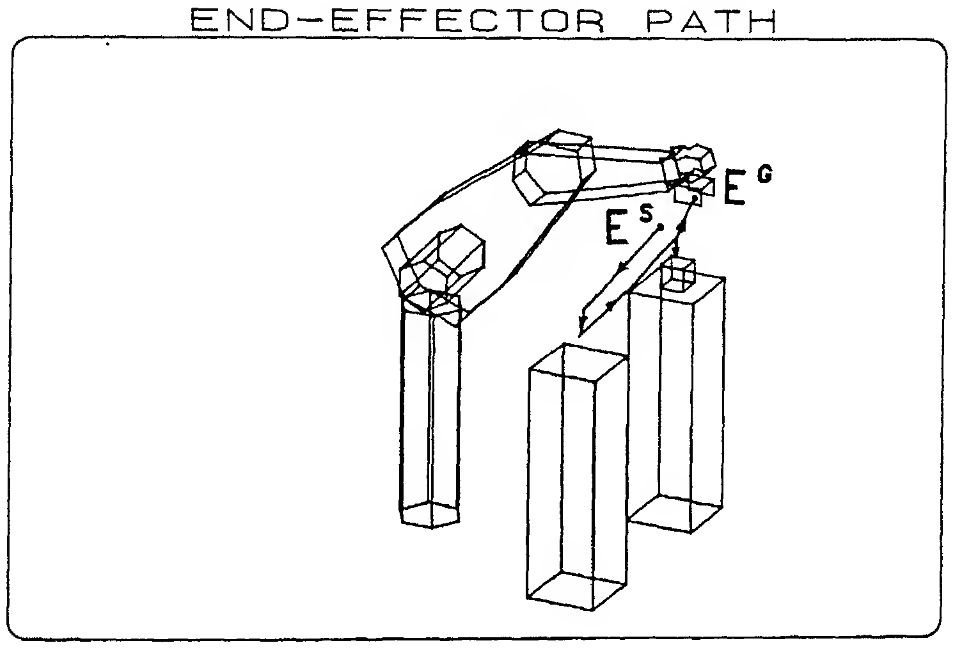


Figure 6.17: Path traced by the end-effector for Example 2

*Example 3: Pick and place operation with one static obstacle*

One static obstacle has been added to the problem of Example 2 to demonstrate the motion of the manipulator through state change operation. Figure 6.18 shows the position of the manipulator just before picking up the object. The obstacle in the environment is in the form of a sheet between the two tables as can be easily deduced from Figure 6.18. Figure 6.19 shows the position of the manipulator

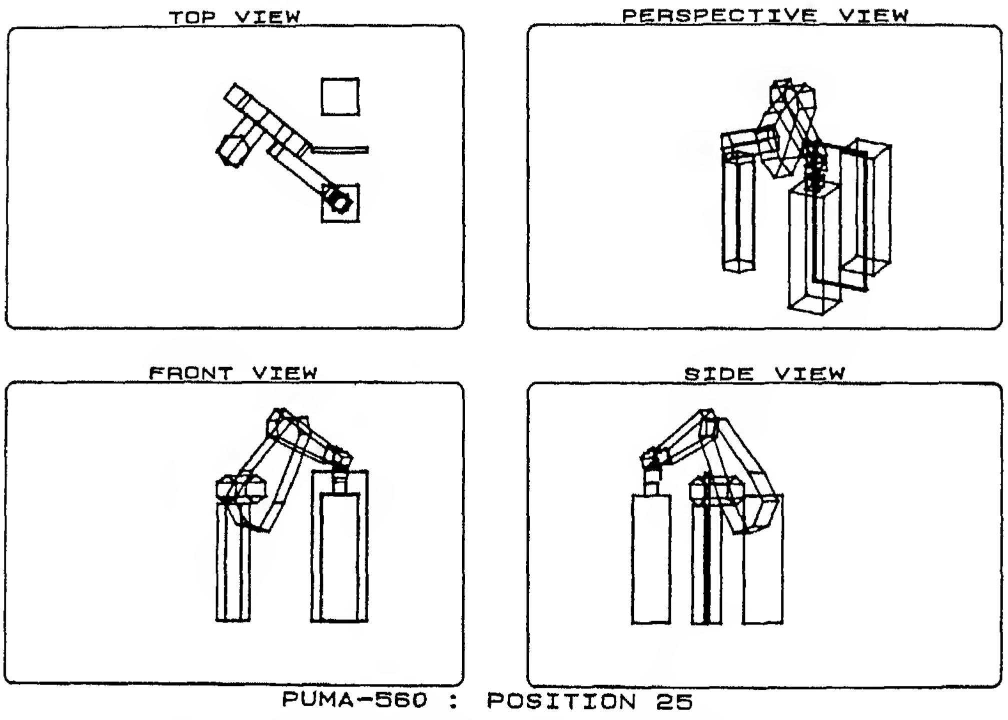


Figure 6.18: Scene of the environment just before picking

when it is midway between the two tables. It is observed that the end-effector has moved up to circumvent the sheet in between. The upward drift has been brought about primarily through *DETOUR* state. It was observed that the manipulator momentarily goes into the *SURRENDER* state in the $117^{th}$ second of execution at position 43 and again at the $127^{th}$ second of execution at position 47. The position in Figure 6.19 is to be compared with the position shown in Figure 6.14 which shows the same instant (approximately) with the sheet absent. In Figure 6.20

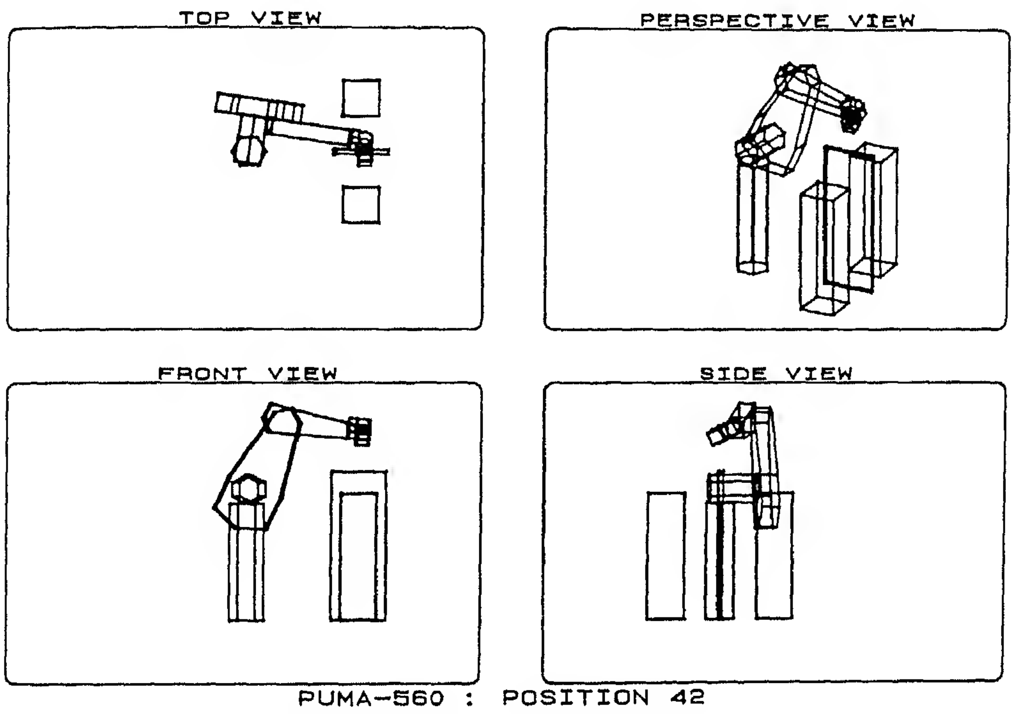


Figure 6.19: Position of the manipulator while it is circumventing the obstacle

the manipulator has reached the approach position of the placing location, and is about to start orienting the gripper. The position just after placing is shown in Figure 6.21. This position is same as that for Figure 6.16 and is given here to indicate the extra time needed for circumventing the obstacle. This position is attained during the 233rd second of execution while it was attained in the 124th second of execution in Example 2. The end-effector path is shown in Figure 6.22 with arrows. It is observed that this path deviates towards the top of the sheet in contrast to the path shown in Figure 6.17. It is to be noted that not much can be attained through *PAUSE* state in this pick-and-place operation, where the path is not so well defined. However, in path-preference problems, the *PAUSE* state will be essential to reconfigure the manipulator. The joint value graphs for the problem of Example 3 are presented in Figures 6.23 and 6.24. The joint velocity graphs are shown in Figures 6.25 and 6.26 while joint torque graphs of Joints 1, 2

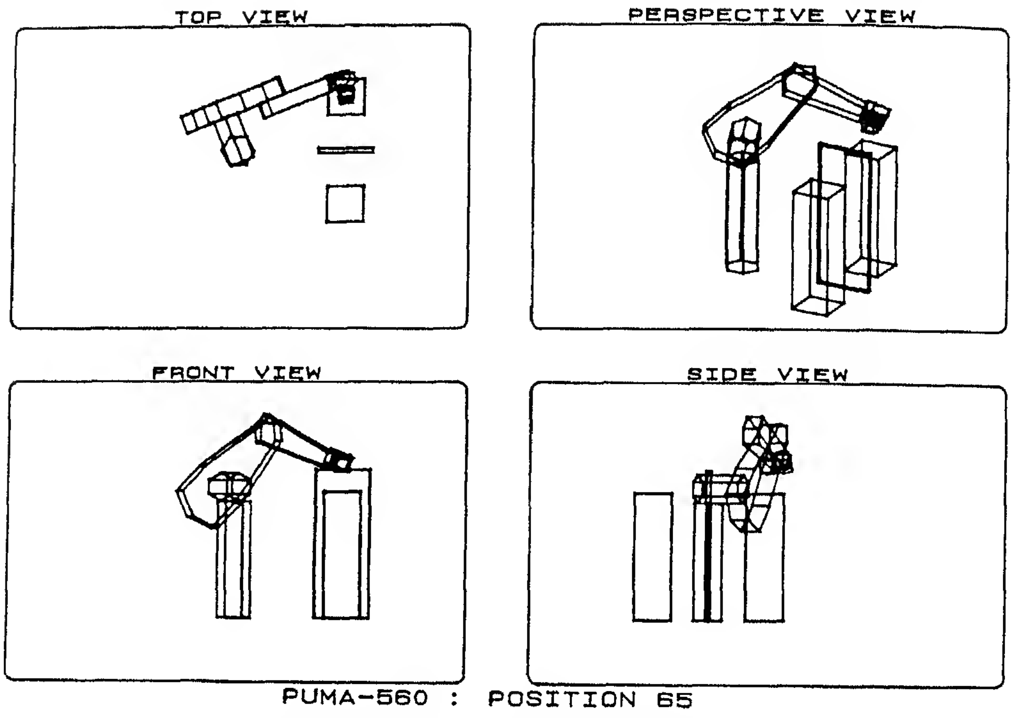


Figure 6.20: Position of the manipulator just before orienting

and 3 are presented in Figures 6.27, 6.28 and 6.29.

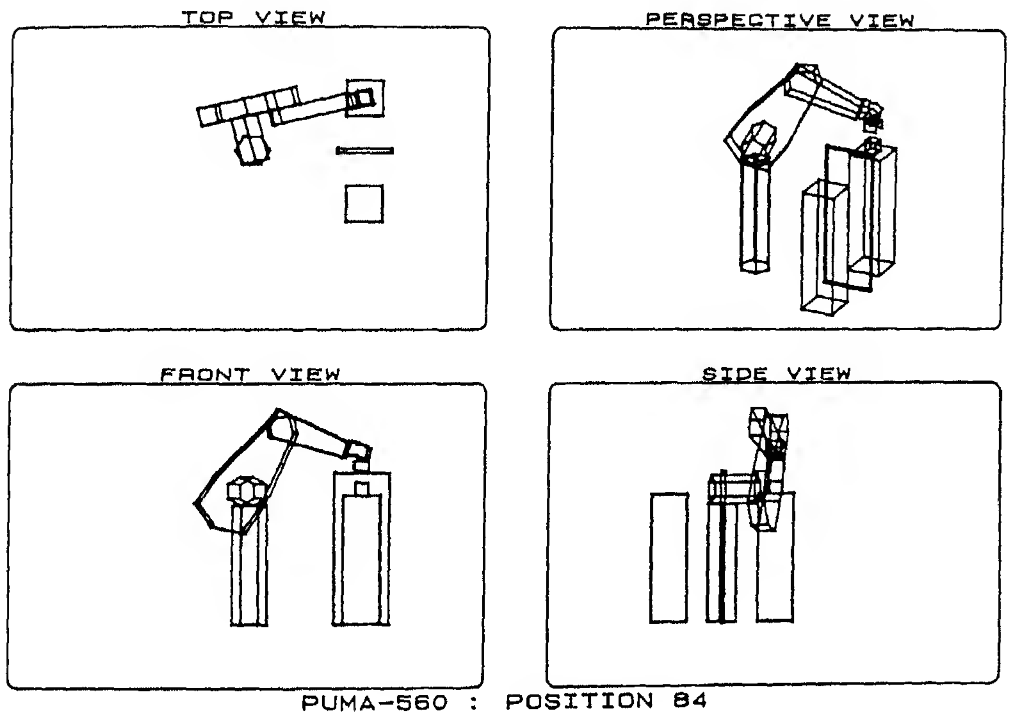


Figure 6.21: Position of the manipulator just after placing the object

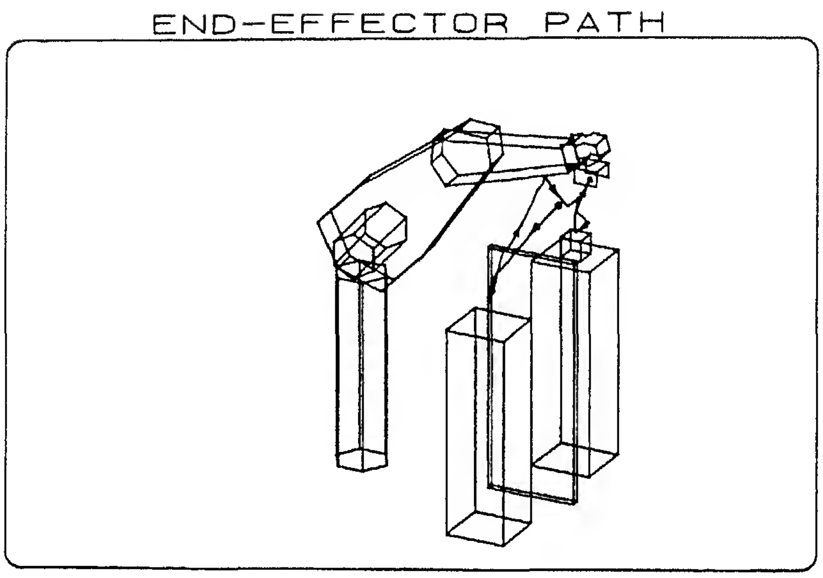


Figure 6.22: Path of end-effector for Example 3

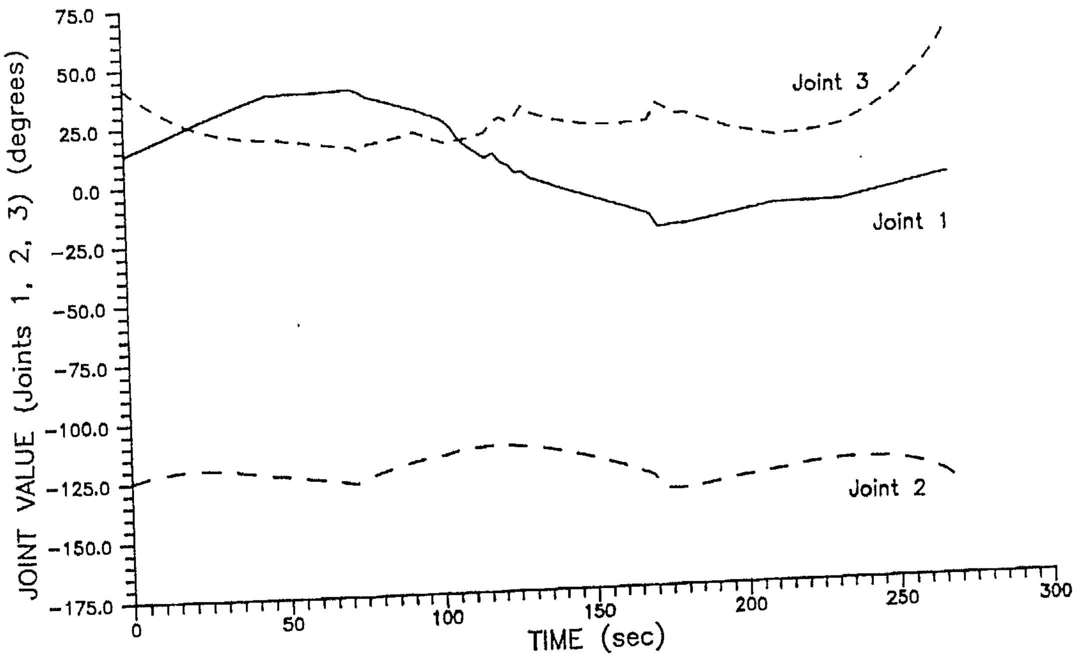


Figure 6.23: Joint value graph of Joints 1, 2 and 3

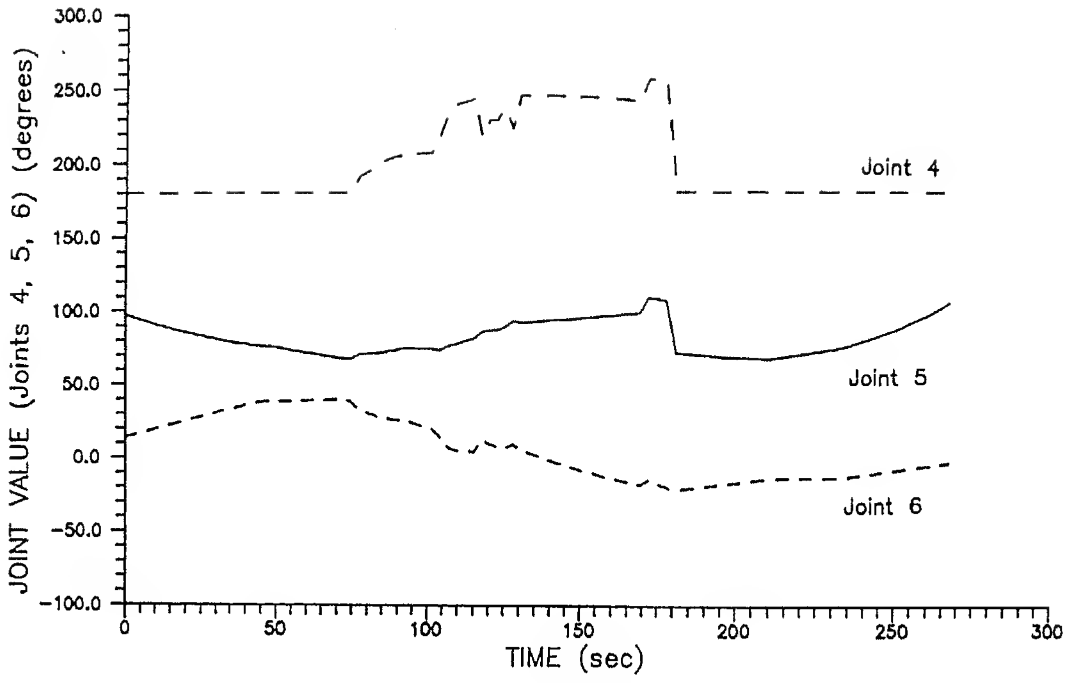


Figure 6.24: Joint value graph of Joints 4, 5 and 6

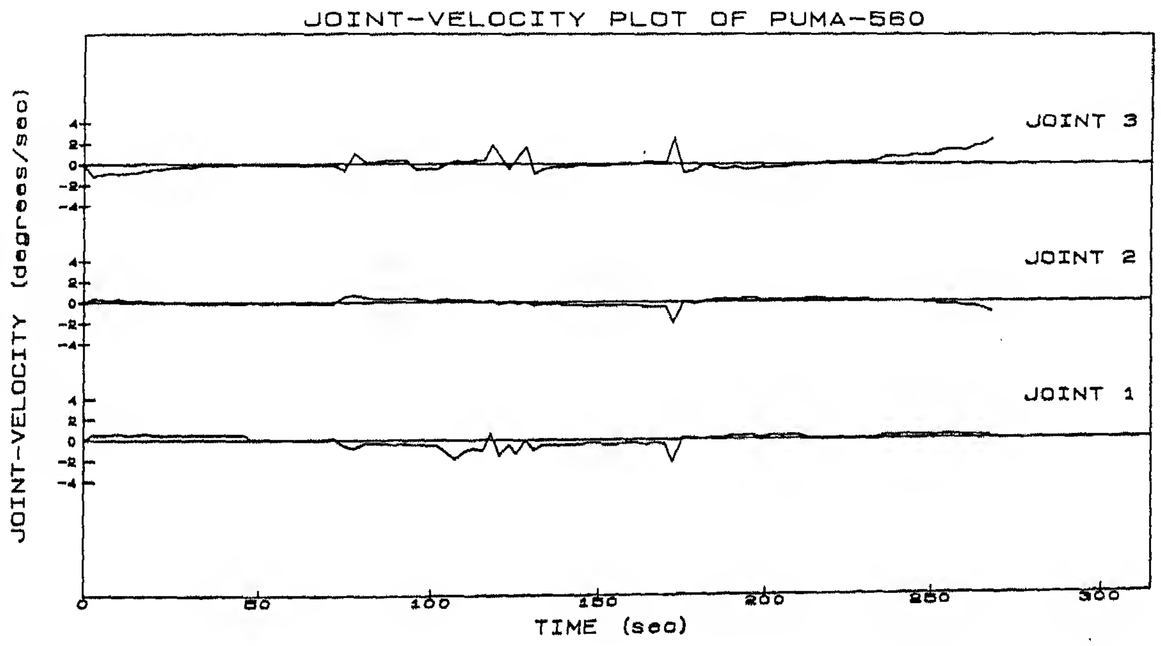


Figure 6.25: Joint velocity graph of Joints 1, 2 and 3

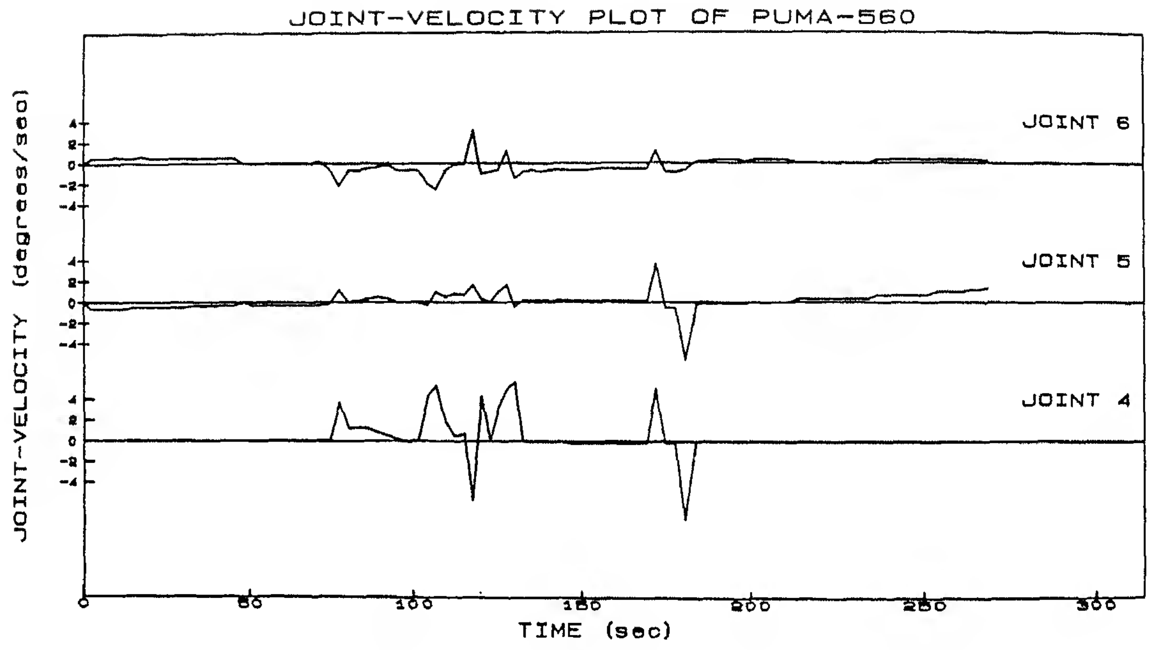


Figure 6.26: Joint velocity graph of Joints 4, 5 and 6

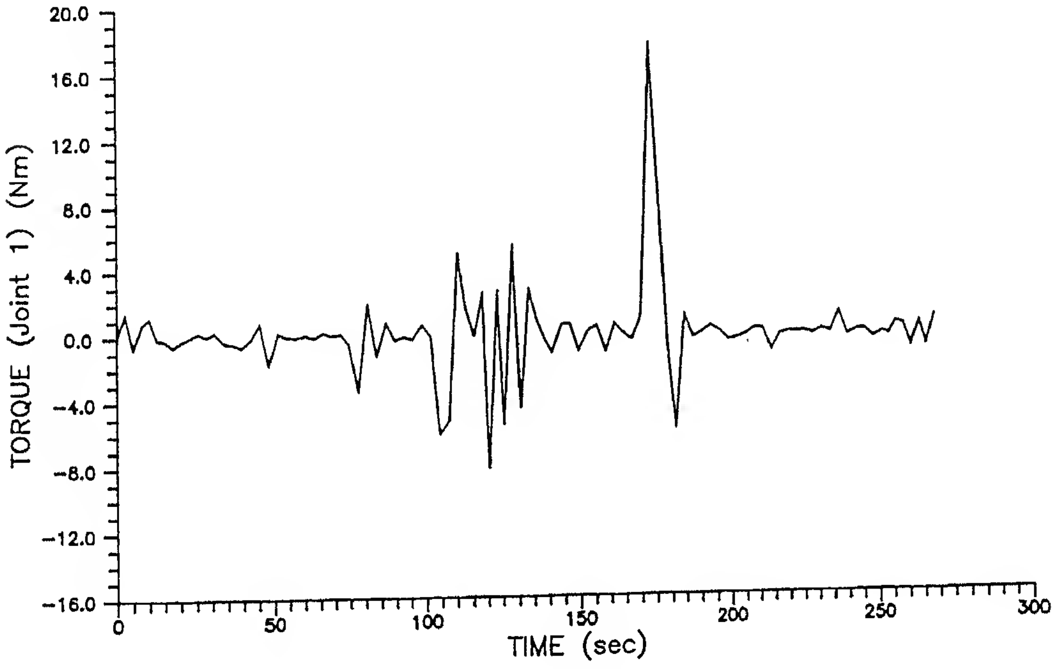


Figure 6.27: Joint torque graph of Joint 1

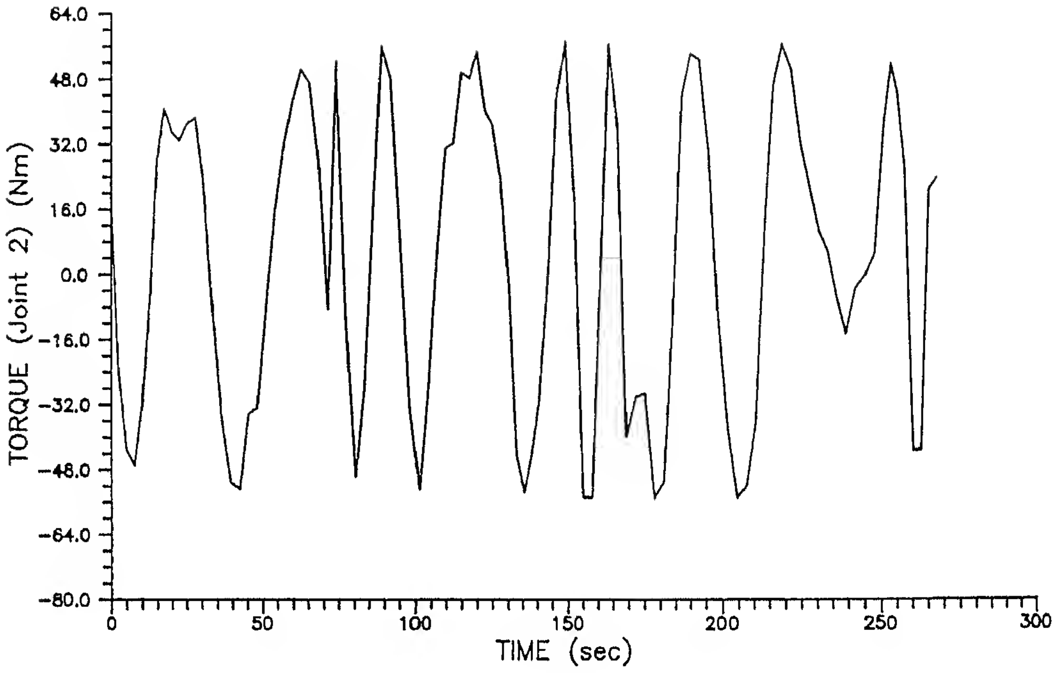


Figure 6.28: Joint torque graph of Joint 2

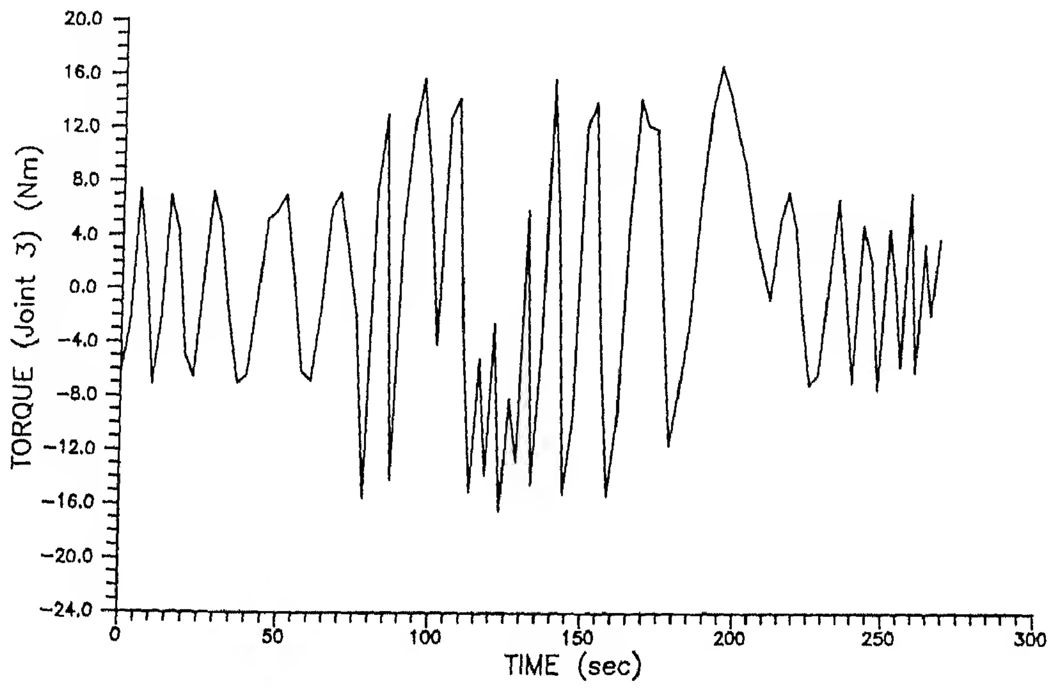


Figure 6.29: Joint torque graph of Joint 3

*Example 4 : Pick-and-place operation with one static and one moving obstacle.* In this example the same problem of Example 3 is solved, with another moving obstacle introduced. The starting position of the manipulator (which is same for all the pick-and-place examples discussed earlier) is shown in Figure 6.30. The

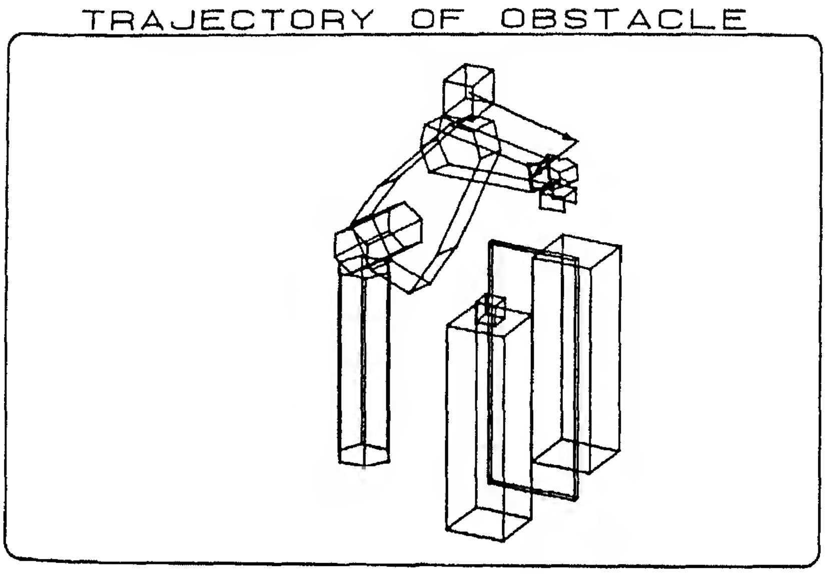


Figure 6.30: Trajectory of the moving obstacle

trajectory of the moving obstacle is shown in the figure with arrows. The obstacle finally positions itself just above the sheet when the manipulator's arm is about to pass through this region. It is to be noted that the above mentioned trajectory of the obstacle has been assigned for simulation purpose only. This trajectory, however is unknown to the task planner.

Figure 6.31 shows the position of the manipulator just before picking up the object. The position of the moving obstacle has changed and it is now seen to approach towards the arm. Figure 6.32 shows the manipulator's position at around the same time of Figure 6.19. It is observer that the manipulator has not yet been

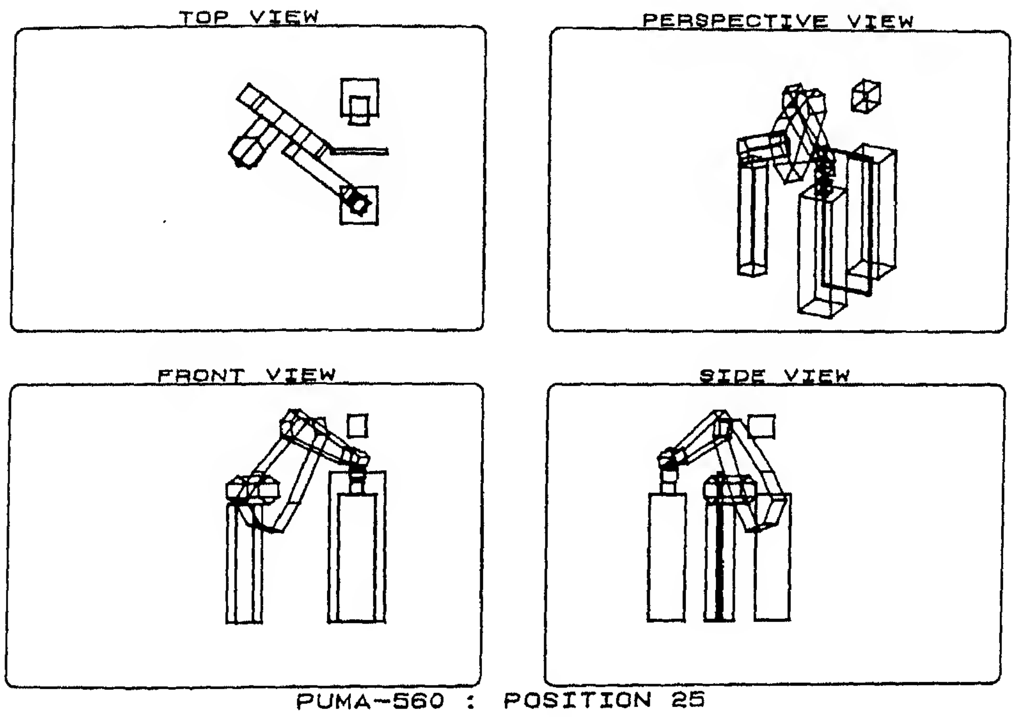


Figure 6.31: Position of manipulator and obstacle just before picking

able to reach midway between the two goal positions as in the earlier examples. This extra time was needed to steer clear of the moving obstacle by being in *SURRENDER* state. Instead, the manipulator is seen to have drifted towards its base from where it starts to approach its goal. Figure 6.33 shows the manipulator's position when it has reached midway between its goal positions. From the FRONT VIEW, it is observed that a collision has been avoided in this position when it is nearest to the obstacles. Figure 6.34 shows the manipulator's configuration when it has steered clear of the obstacles while Figure 6.35 shows the position just after placing the object. The entire path traced by the end-effector is shown in Figure 6.36 with arrow lines which also shows the manipulator and the moving obstacle in its final parking position. The joint value graphs for the problem of Example 4 are presented in Figures 6.37 and 6.38. The joint velocity graphs are shown in Figures 6.39 and 6.40 while joint torque graphs of Joints 1, 2 and 3 are

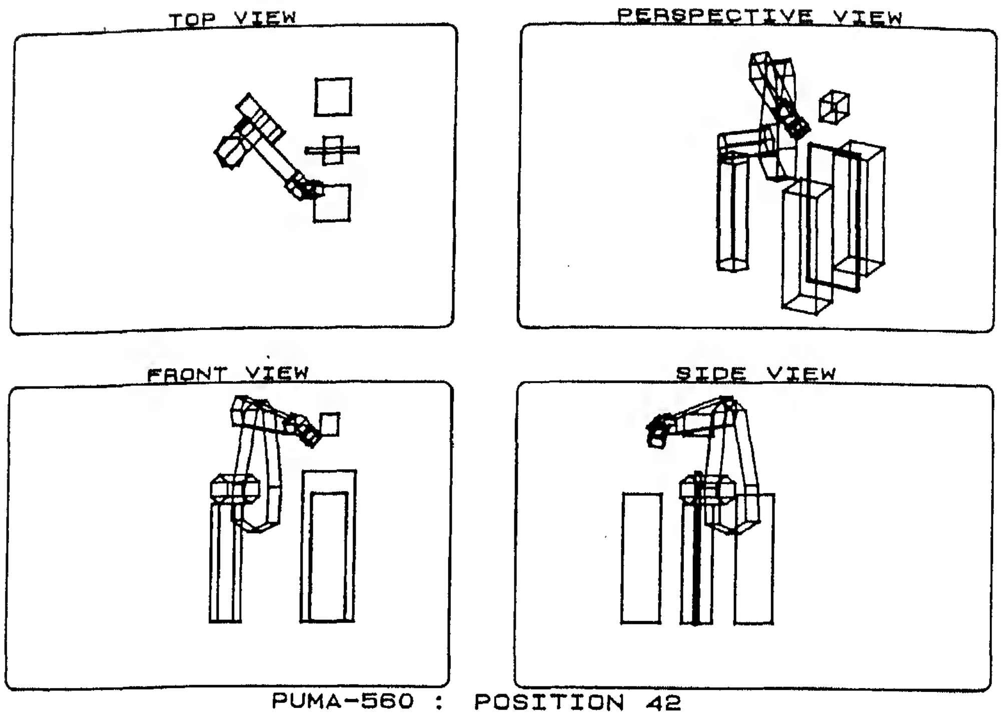


Figure 6.32: Position of manipulator after stabilizing from *SURRENDER* state

presented in Figures 6.41, 6.42 and 6.43.

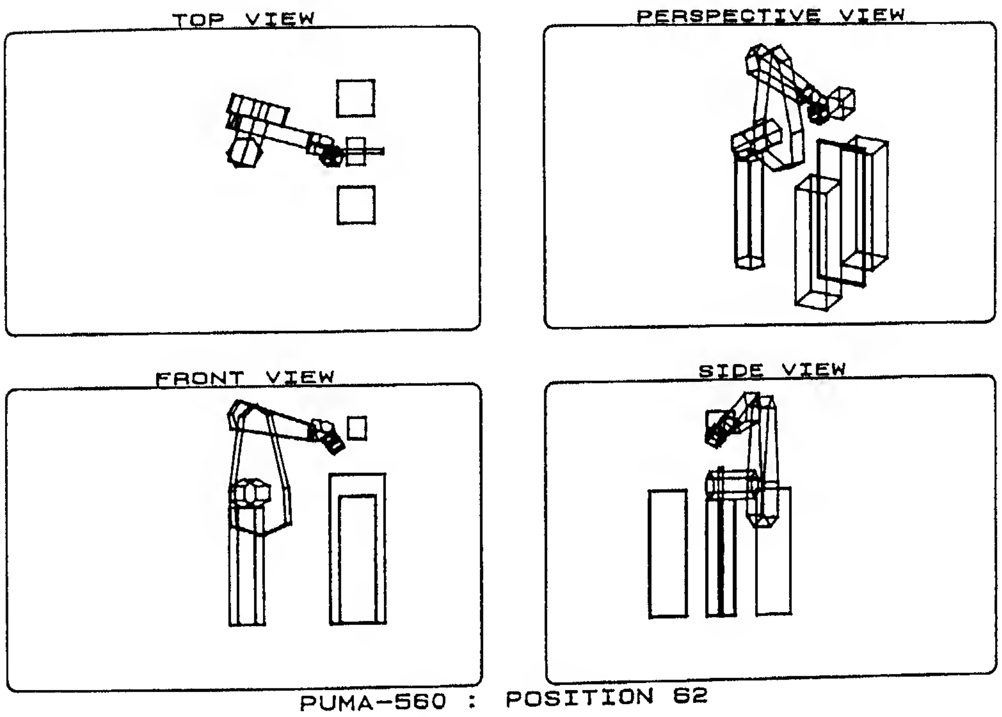


Figure 6.33: The manipulator when it is nearest to the obstacles

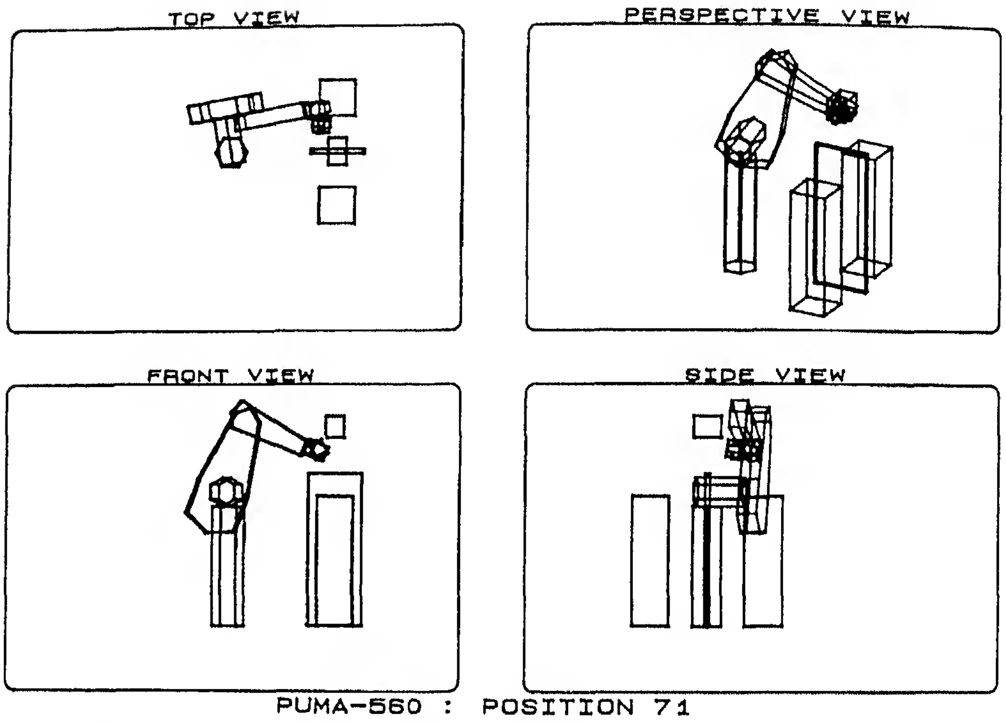


Figure 6.34: Position of manipulator after circumventing obstacles

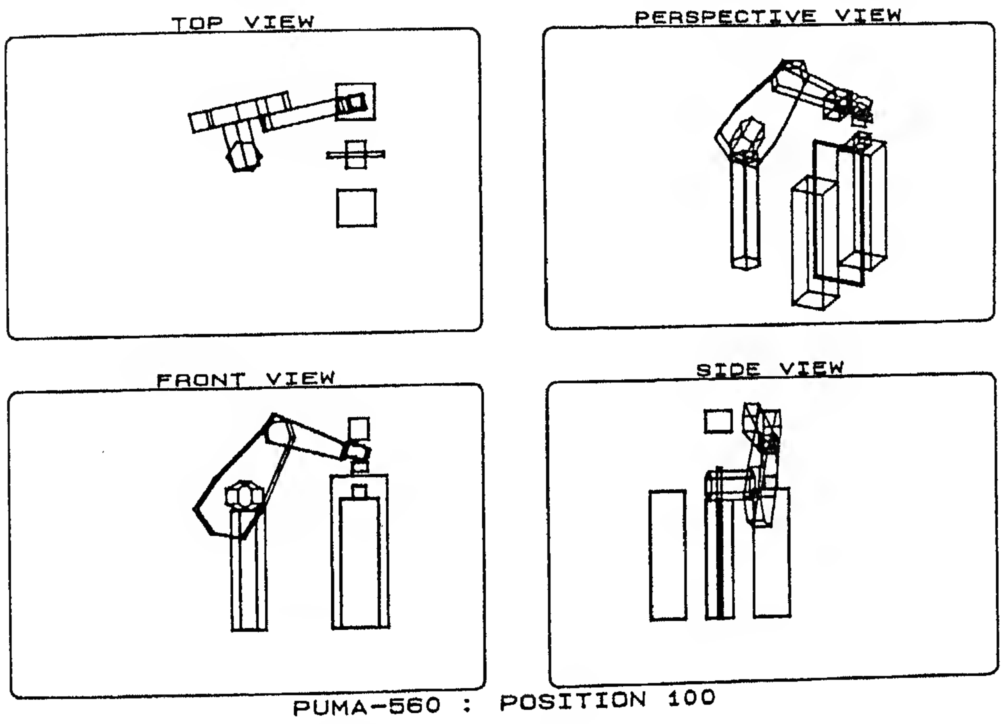


Figure 6.35: Position of manipulator after placing the block

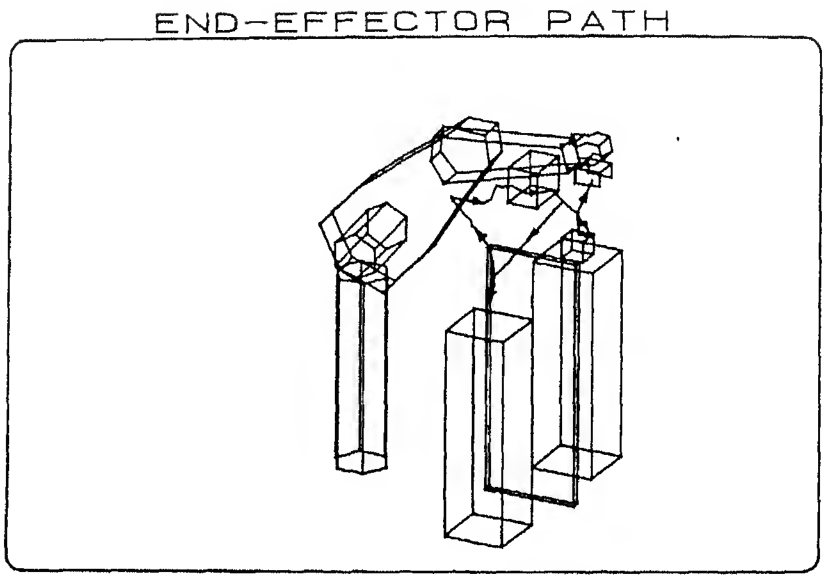


Figure 6.36: End-effector path for Example 4

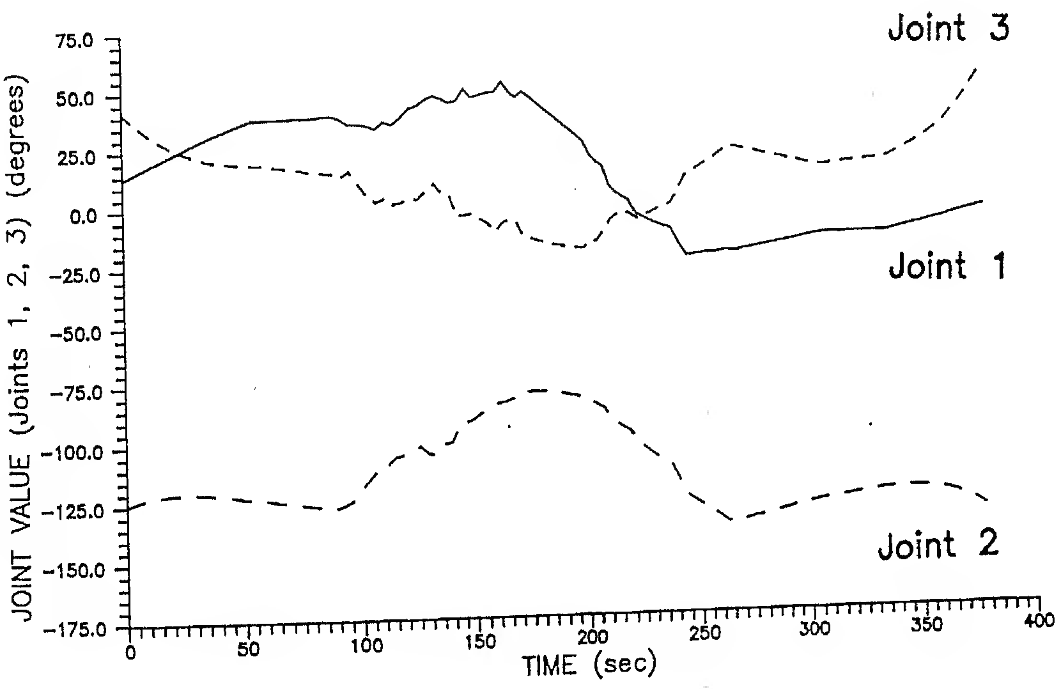


Figure 6.37: Joint value graph of Joints 1, 2 and 3

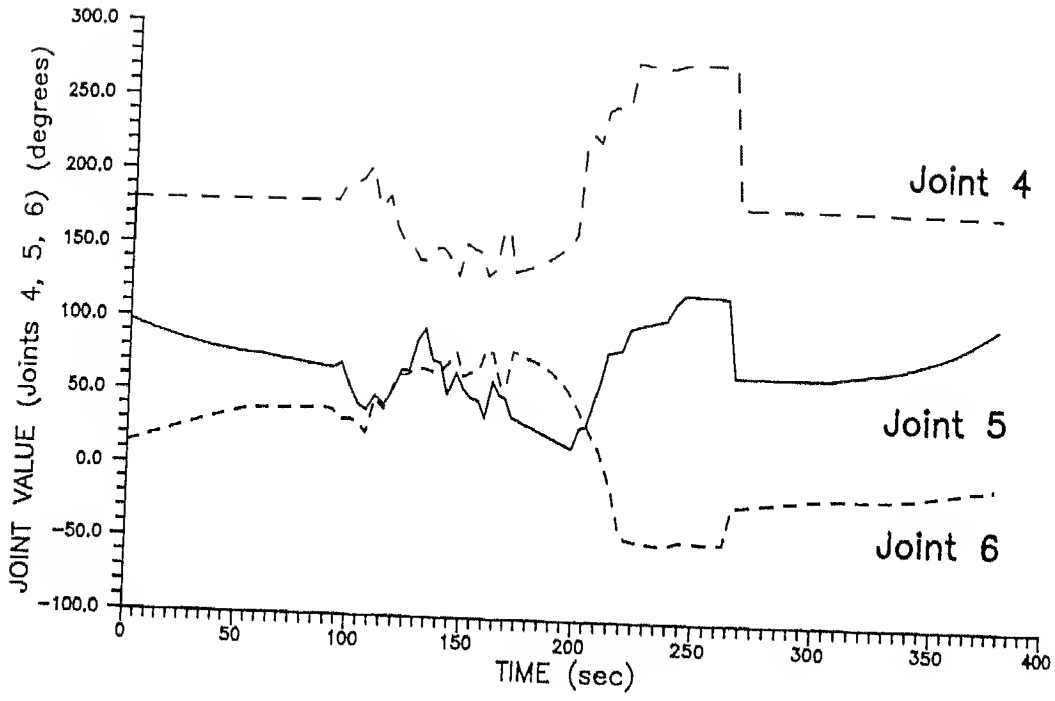


Figure 6.38: Joint value graph of Joints 4, 5 and 6

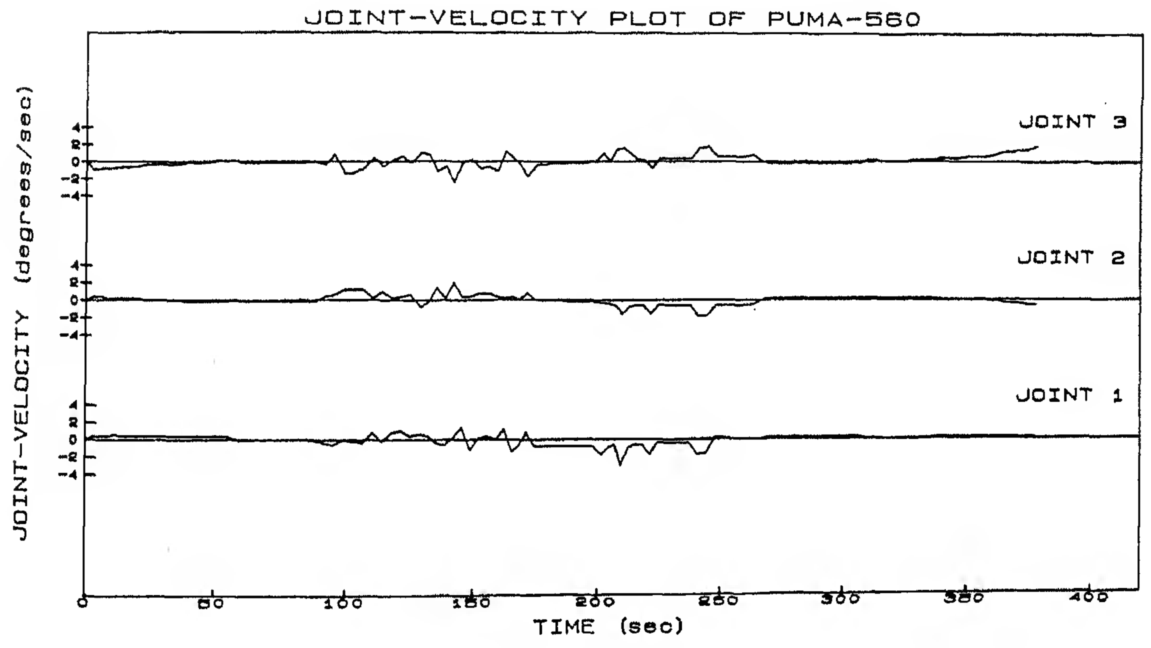


Figure 6.39: Joint velocity graph of Joints 1, 2 and 3

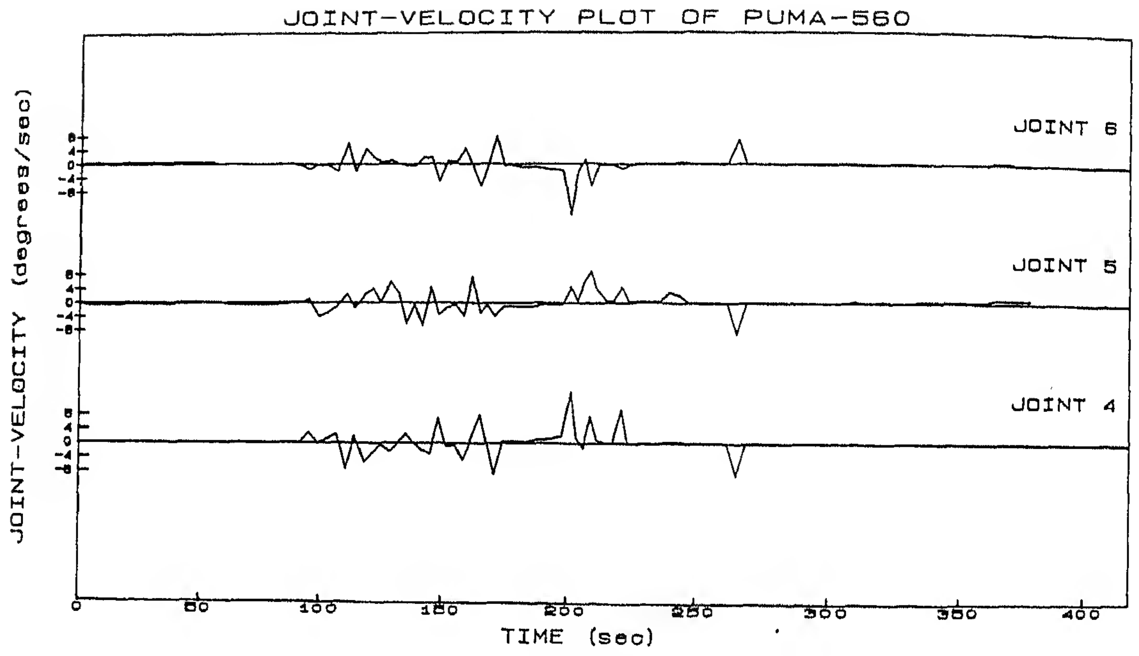


Figure 6.40: Joint velocity graph of Joints 4, 5 and 6

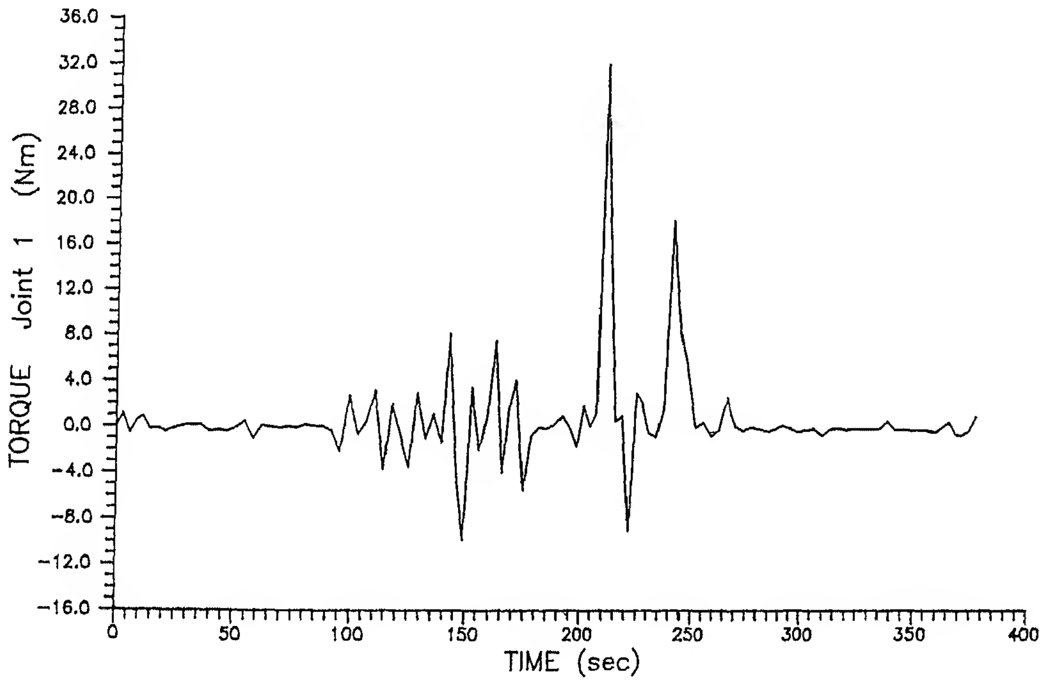


Figure 6.41: Joint torque graph of Joint 1

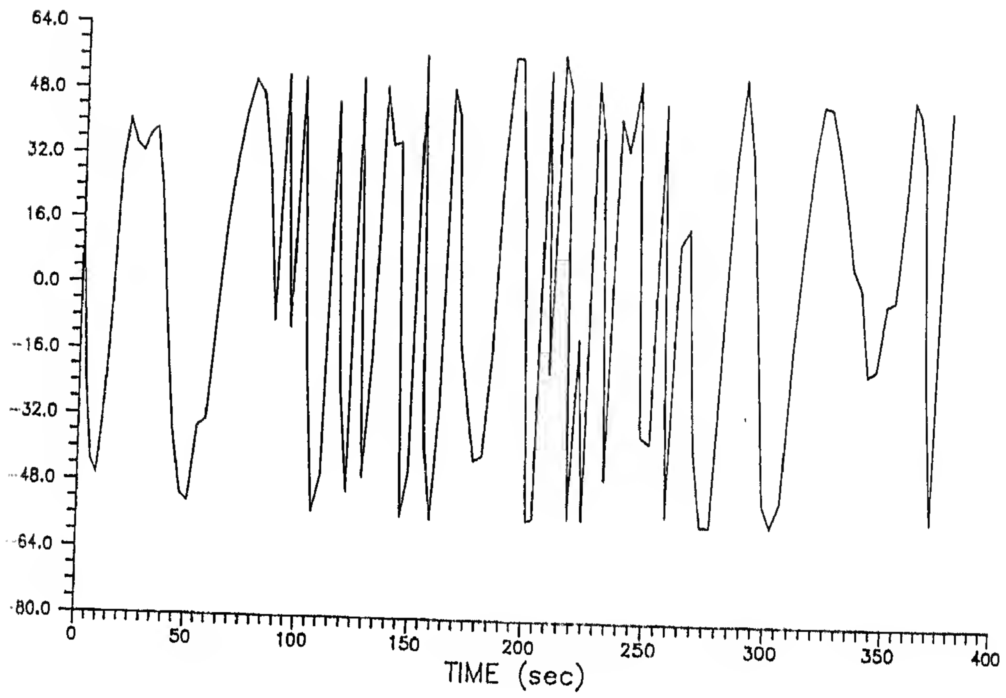


Figure 6.42: Joint torque graph of Joint 2

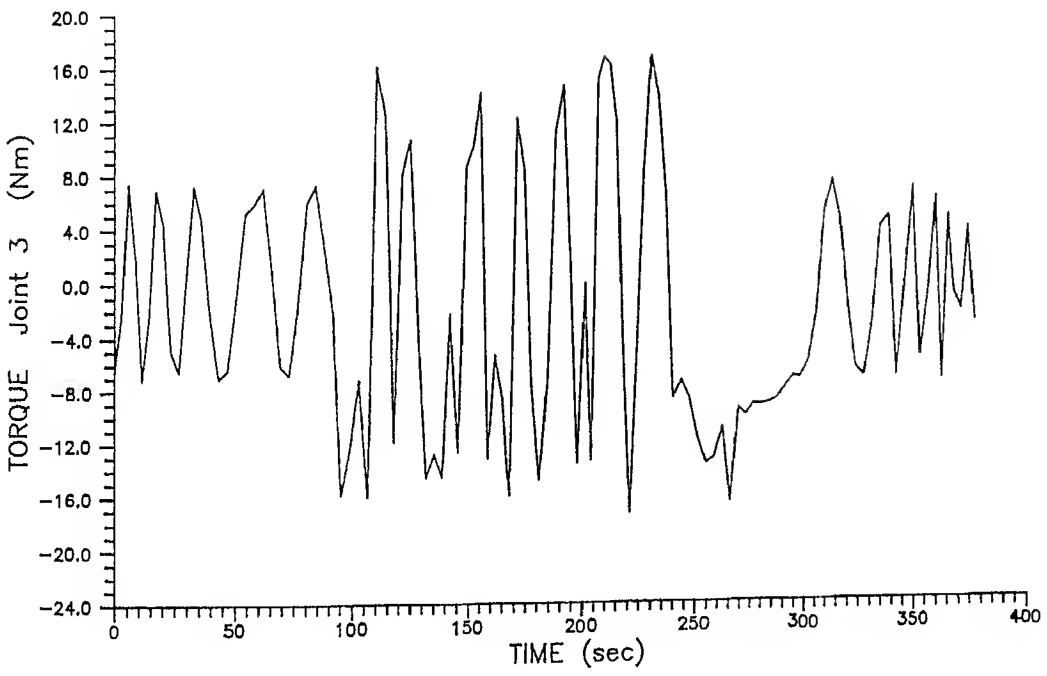


Figure 6.43: Joint torque graph of Joint 3

Finally, a table is presented which gives the *sequence* in which the state of the manipulator and the states of the gripper have changed in the problem of Example 4. The table lists the associated execution time and the position coordinates of the end-effector in the base frame, namely $(x_F, y_F, z_F)$. The execution time also appears in the table at each step. The table appears in pages 241 and 242.

The execution time for the various problems solved in this chapter are tabulated in Table 6.4. It is observed that the execution time increases with the number of bodies used in simulation.

| Sequence of state change operation | | | | | | |
|---|---|---|---|---|---|---|
| Step No. | Manipulator State | Gripper State | Time (s) | $x_F$ | $y_F$ | $z_F$ |
| 1 | NATURAL | EMPTY | 0.0000 | -699 | 0 | 299 |
| 2 | NATURAL | EMPTY | 2.9100 | -693 | -15 | 290 |
| 3 | NATURAL | EMPTY | 5.8200 | -687 | -30 | 281 |
| ... | ... | ... | ... | ... | ... | ... |
| 18 | NATURAL | EMPTY | 54.4900 | -609 | -266 | 143 |
| 19 | ORIENTING | EMPTY | 58.2200 | -605 | -283 | 133 |
| 20 | APPROACHING | EMPTY | 61.9600 | -605 | -283 | 132 |
| ... | ... | ... | ... | ... | ... | ... |
| 27 | APPROACHING | EMPTY | 88.0500 | -602 | -294 | 65 |
| 28 | NATURAL | HOLDING | 91.7800 | -600 | -299 | 57 |
| 29 | DETOUR | HOLDING | 95.4100 | -605 | -272 | 103 |
| ... | ... | ... | ... | ... | ... | ... |
| 32 | DETOUR | HOLDING | 106.5600 | -580 | -235 | 170 |
| 33 | SURRENDER | HOLDING | 110.3500 | -575 | -232 | 181 |
| 34 | DETOUR | HOLDING | 114.1900 | -562 | -221 | 204 |
| 35 | SURRENDER | HOLDING | 117.8700 | -556 | -220 | 213 |
| ... | ... | ... | ... | ... | ... | ... |
| 39 | SURRENDER | HOLDING | 131.9900 | -514 | -244 | 282 |
| 40 | DETOUR | HOLDING | 135.3900 | -512 | -244 | 283 |
| 41 | NATURAL | HOLDING | 138.7400 | -512 | -243 | 281 |
| 42 | DETOUR | HOLDING | 142.0900 | -498 | -222 | 290 |
| 43 | SURRENDER | HOLDING | 145.6600 | -474 | -227 | 309 |
| 44 | DETOUR | HOLDING | 148.9000 | -472 | -225 | 310 |
| ... | ... | ... | ... | ... | ... | ... |
| 47 | DETOUR | HOLDING | 158.7400 | -452 | -219 | 327 |
| 48 | SURRENDER | HOLDING | 162.0900 | -430 | -218 | 343 |
| 49 | DETOUR | HOLDING | 165.2700 | -423 | -214 | 350 |
| 50 | NATURAL | HOLDING | 168.0700 | -423 | -214 | 349 |
| 51 | DETOUR | HOLDING | 171.1500 | -429 | -197 | 341 |
| 52 | NATURAL | HOLDING | 174.5000 | -429 | -197 | 340 |
| ... | ... | ... | ... | ... | ... | ... |

Table 6.3: The sequence of state change operation : Part 1. The coordinates of the end-effector in the base frame are given as $(x_F, y_F, z_F)$.

| Sequence of state change operation | | | | | | |
|---|---|---|---|---|---|---|
| Step No. | Manipulator State | Gripper State | Time (s) | $x_F$ | $y_F$ | $z_F$ |
| 59 | NATURAL | HOLDING | 197.9500 | -471 | -78 | 284 |
| 60 | DETOUR | HOLDING | 201.0800 | -470 | -54 | 282 |
| 61 | NATURAL | HOLDING | 203.8300 | -470 | -54 | 281 |
| 62 | DETOUR | HOLDING | 206.6900 | -472 | -31 | 279 |
| 63 | DETOUR | HOLDING | 209.4300 | -466 | 2 | 297 |
| 64 | DETOUR | HOLDING | 212.2300 | -465 | 20 | 305 |
| 65 | NATURAL | HOLDING | 215.0900 | -465 | 19 | 304 |
| 66 | NATURAL | HOLDING | 218.0500 | -472 | 33 | 294 |
| 67 | DETOUR | HOLDING | 221.1900 | -480 | 89 | 278 |
| 68 | NATURAL | HOLDING | 224.0400 | -479 | 88 | 278 |
| ... | ... | ... | ... | ... | ... | ... |
| 72 | NATURAL | HOLDING | 236.7300 | -507 | 140 | 238 |
| 73 | DETOUR | HOLDING | 240.3000 | -520 | 223 | 210 |
| 74 | DETOUR | HOLDING | 243.9800 | -525 | 290 | 189 |
| 75 | NATURAL | HOLDING | 247.6000 | -524 | 289 | 189 |
| ... | ... | ... | ... | ... | ... | ... |
| 79 | NATURAL | HOLDING | 262.2700 | -580 | 294 | 141 |
| 80 | ORIENTING | HOLDING | 265.9500 | -594 | 297 | 129 |
| 81 | APPROACHING | HOLDING | 269.5800 | -603 | 393 | 1 |
| ... | ... | ... | ... | ... | ... | ... |
| 91 | APPROACHING | HOLDING | 305.7700 | -600 | 313 | 44 |
| 92 | RETRACTING | EMPTY | 309.4000 | -600 | 306 | 50 |
| ... | ... | ... | ... | ... | ... | ... |
| 99 | RETRACTING | EMPTY | 334.6600 | -597 | 302 | 108 |
| 100 | NATURAL | EMPTY | 338.3400 | -598 | 301 | 116 |
| ... | ... | ... | ... | ... | ... | ... |
| 112 | NATURAL | EMPTY | 377.2300 | -694 | 205 | 291 |

Table 6.3: The sequence of state change operation (contd.) : Part 2

| *Description* | *Example* | *Bodies* | *Execution time per step (sec)* |
|---|---|---|---|
| Path-Preference | 1 | 1 | 2.668 |
| Point-to-Point | 2 | 0 | 1.758 |
| Point-to-Point | 3 | 1 | 2.759 |
| Point-to-Point | 4 | 2 | 3.368 |

Table 6.4: Execution time for various examples solved in this chapter

## 6.6 Object-Oriented Computer Simulation

Computer simulation of the PUMA-560 requires modelling the robot and its environment and building graphic modules for viewing the robot in 3-D space. This section describes the modelling techniques and the constructs for mathematical structures that are required for appropriate representation of the robot equations.

The present implementation uses *object-oriented* style of programming. Important advantages gained by this approach are extensibility and easy maintenance of programs. Future programmers and users need not be concerned with the complete details of implementation; rather they need to modify only the appropriate "objects" that need improvement. First the "objects" are described and then the interaction between the objects is established[2]. The header files for the C++ implementation appear in Appendices B thru E and are referred whenever necessary.

The "objects" can be widely classified into four categories.

1. Objects for modelling mathematical classes
2. Objects for modelling the environment
3. Objects for modelling the robot
4. Objects for graphic display

Each of the categories are considered separately and their characteristics are briefly described.

### 6.6.1 Objects for modelling mathematical classes

The mathematical classes of interest are

1. points in 2-D space
2. points in 3-D space
3. vectors (generally in 3-D space)

[2]It is to be noted that here the "objects" encompass the mathematical as well as the physical constructs.

4. planes in 3-D space

5. transformations

6. fuzzy numbers

Along with the definitions of objects some associated functions which need special explanation are described.

### Points

Objects describing points in 2-D and 3-D space are standard and hence are not described here in detail. Their descriptions are given in Section B.1.

### Vector

A vector is a rudimentary mathematical object required for representing matrix operations and for solving the inverse dynamics problem. The necessary operations on vectors are cross product, dot product, finding magnitude, scalar multiplication and finding angle between two vectors[3]. The description of a vector object is given in Section B.2. Most of the vector functions defined in `class Vector` can be implemented easily. The one which deserves a little explanation is on rotation of a unit vector, which is described below.

Let $\vec{v}_1$ be a unit vector that needs to be rotated by an angle $\lambda$ towards another unit vector $\vec{v}_2$ as shown in Figure 6.44. Let the resulting vector be $\vec{v}_3$. Define a vector $\vec{v}_4$ which is perpendicular to both $\vec{v}_1$ and $\vec{v}_2$ as

$$\vec{v}_4 = \vec{v}_1 \times \vec{v}_2.$$

Since $\vec{v}_4$ and $\vec{v}_3$ are perpendicular $\vec{v}_3 \cdot \vec{v}_4 = 0$ which leads to the following equation.

$$(v_3)_x(v_4)_x + (v_3)_y(v_4)_y + (v_3)_z(v_4)_z = 0 \tag{6.10}$$

Further, since the angle between $\vec{v}_3$ and $\vec{v}_1$ is known to be $\lambda$ it is known that

$$\cos\lambda = \frac{\vec{v}_3 \cdot \vec{v}_1}{\mid \vec{v}_1 \mid \mid \vec{v}_3 \mid}$$

[3]All these standard operations may be obtained from elementary books of mathematics [127]

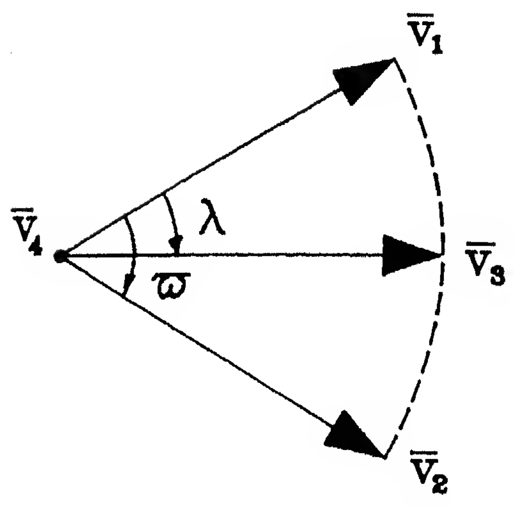


Figure 6.44: Rotation of a unit vector

which leads to

$$(v_1)_x(v_3)_x + (v_1)_y(v_3)_y + (v_1)_z(v_3)_z = \cos\lambda \tag{6.11}$$

Similarly, $\vec{v}_3$ may also be written as

$$(v_2)_x(v_3)_x + (v_2)_y(v_3)_y + (v_2)_z(v_3)_z = \cos(\varpi - \lambda) \tag{6.12}$$

One can find $\vec{v}_3$ by solving (6.10), (6.11) and (6.12). However this method of finding $\vec{v}_3$ gives rise to numerical difficulties since it involves a lot of divisions, and has not been found suitable for implementation. Therefore, a simple but approximate solution has been adopted which is based on geometry, but numerically stable. Figure 6.45 shows the same two vectors $\vec{v}_1$ and $\vec{v}_2$ and it is required to rotate vector $\vec{v}_1$ by an angle $\lambda$. First the vector $\vec{v}_2 - \vec{v}_1$ is found and along that vector a distance of $2\sin(\lambda/2)$ is cut along its length starting from the tip of $\vec{v}_1$. Let this point along $\vec{v}_2 - \vec{v}_1$ be represented as $V_3$. Then the line from the base of $\vec{v}_1$ to $V_3$ is approximately equal to $\vec{v}_3$. This solution tends to the exact solution if $\lambda$ is a small number. In the example runs undertaken, it was found that $\lambda \leq 30°$.

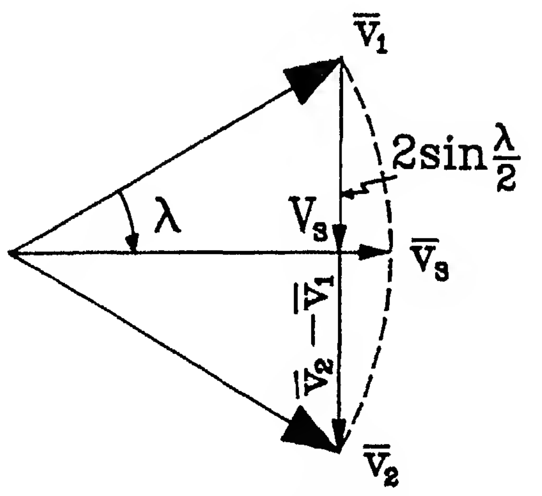


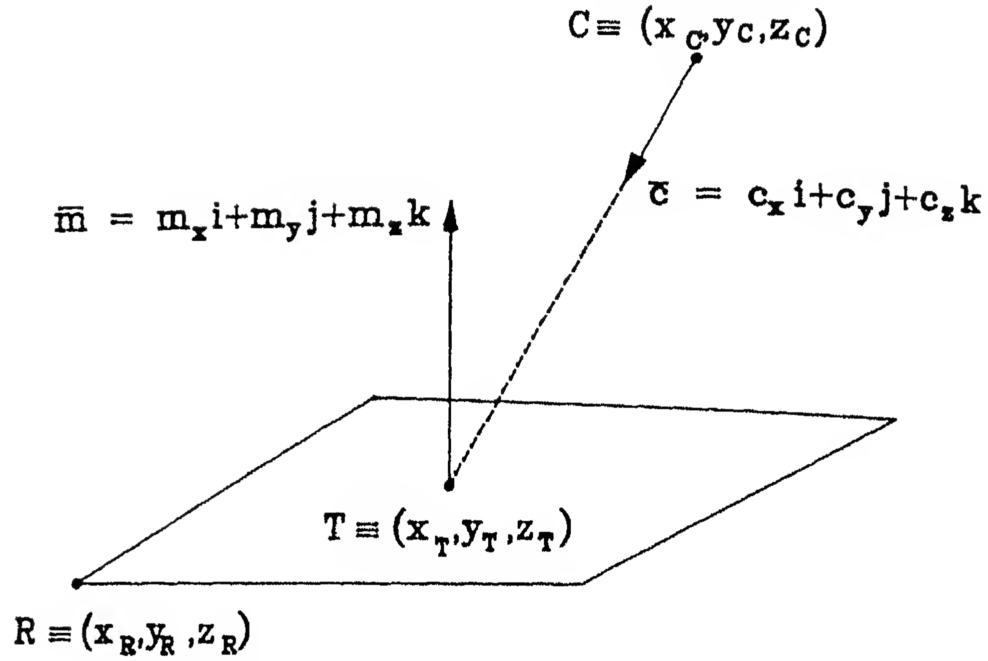
Figure 6.45: Approximate rotation of a unit vector

**Plane**

The representation of a *plane* is necessary to model the links and the obstacles. Normally it is sufficient to describe a plane by specifying 3 points in space. Since all planes encountered in the implementation are rectangles, the planes are represented by a set of four points which form the vertices of the rectangle. Since the four points are known to be coplanar, such a representation reduces computation time. The object describing `class Plane` appears in Section B.3. The normal vector to the plane is found by ordering the 4 points of the plane according to a fixed convention. If A,B,C and D, the four points of a plane, are specified in anti-clock wise direction, then the outward normal vector is defined by $\vec{n} = \vec{AB} \times \vec{AD}$.

An approach to find out the point of intersection of a ray (i.e. a vector with a known base) with a given plane is now presented. The ray is defined by the vector $\vec{c} = c_x\hat{\imath} + c_y\hat{\jmath} + c_z\hat{k}$ emanating from point $C \equiv (x_C, y_C, z_C)$ (See Figure 6.46). Further, let $\vec{m} = m_x\hat{\imath} + m_y\hat{\jmath} + m_z\hat{k}$ define the normal to a plane and let $R \equiv (x_R, y_R, z_R)$ be a known point on the plane. The point of intersection can be found out as follows.

The equation of the plane is written with the help of one of the known points



$C \equiv (x_C, y_C, z_C)$
$\bar{c} = c_x i + c_y j + c_z k$
$\bar{m} = m_x i + m_y j + m_z k$
$T \equiv (x_T, y_T, z_T)$
$R \equiv (x_R, y_R, z_R)$

Figure 6.46: Finding the intersection of a ray with a plane

(e.g., any one of the corners), say $R \equiv (x_R, y_R, y_R)$. Thus

$$m_x x + m_y y + m_z z = m_x x_R + m_y y_R + m_z z_R \tag{6.13}$$

The equation of line $CT$ is written as

$$\frac{x - x_C}{c_x} = \frac{y - y_C}{c_y} = \frac{z - z_C}{c_z} \tag{6.14}$$

Let the point of intersection be represented by $T \equiv (x_T, y_T, z_T)$. Then point $T$ satisfies both equations (6.13) and (6.14) and thus

$$m_x x_T + m_y y_T + m_z z_T = m_x x_R + m_y y_R + m_z z_R \tag{6.15}$$

$$\frac{x_T - x_C}{c_x} = \frac{y_T - y_C}{c_y} = \frac{z_T - z_C}{c_z} \tag{6.16}$$

Solving the above set yields the coordinates of point $T$ as

$$\begin{aligned} x_T &= c_x \bar{t} + x_C \\ y_T &= c_y \bar{t} + y_C \\ z_T &= c_z \bar{t} + z_C \end{aligned} \tag{6.17}$$

where

$$\bar{t} = \frac{m_x(x_R - x_C) + m_y(y_R - y_C) + m_z(z_R - z_C)}{\vec{m} \cdot \vec{c}}.$$

### Transformation

The object `Transformation` deals with $4 \times 4$ $A$-matrices given in (A.1) or (A.2). The description of `class Transformation` is given in Section B.4. There is a special type of assignment `GripObjectUpdate()` which is needed for grasp objects. This assignment is explained later in Section 6.6.3.

### Fuzzy number

A detailed discussion on fuzzy numbers and fuzzy logic has been given Chapter 2. The computer representation of `class Fuzzy` appears in Section B.5.

## 6.6.2 Objects for modelling the environment

The objects that comprise the environment or interact directly with the environment are

1. path in space
2. obstacles
3. sensor

### Path

A path is considered a spatial curve segment whose general equation is given by

$$x = h_1(t) \qquad y = h_2(t) \qquad z = h_3(t) \tag{6.18}$$

where $h_1$, $h_2$ and $h_3$ are differentiable functions. As parameter $t$ varies, the locus of point $H \equiv (x, y, z)$ is a curve in space. The tangent to a path is calculated by evaluating the quantity

$$\vec{T} = \frac{\mathrm{d}\vec{R}}{\mathrm{d}s} \tag{6.19}$$

where $s$ is the arc length along the curve and $\vec{R}$ is the vector

$$\vec{R} = x\hat{\imath} + y\hat{\jmath} + z\hat{k}$$

In the examples undertaken in this thesis, only straight line segments have been used. The equation of a straight line from point F $\equiv (x_F, y_F, z_F)$ to another point $G \equiv (x_G, y_G, z_G)$ is represented in parametric form by the functions

$$\begin{aligned} h_1(t) &= x_F + t(x_G - x_F) \\ h_2(t) &= y_F + t(y_G - y_F) \\ h_3(t) &= z_F + t(z_G - z_F) \end{aligned} \tag{6.20}$$

The approach for finding a tangent to a path may be obtained from any standard book of mathematics [127].

This section now describes a method for approximately determining an appropriate approach vector $\vec{\mathbf{a}}$ to a given point on a path. The basic criterion used for determining the hand vectors $[\vec{\mathbf{n}}, \vec{\mathbf{s}}, \vec{\mathbf{a}}]$ is that the approach vector should point towards the base of the robot for easy accessibility. For any given path, a direction can be found which is normal to the path as well as subtends the minimum angle towards the base. This direction will approximately decide the easiest approach to the point on the path. Strictly speaking, the accessibility to a point is manipulator dependent, and the following approach is not correct, but it is presented here because it is easy to comprehend and has been found to work well for the simulation runs. Figure 6.47 shows a path from $E^S$ to $E^G$. It is desired to find an appropriate approach vector $\overline{\mathrm{WF}}$ when the end-effector is at F.

Let us denote the path from $E^S$ to $E^G$ by the parametric equation

$$\vec{g}(t) = g_1(t)\hat{\imath} + g_2(t)\hat{\jmath} + g_3(t)\hat{k} \tag{6.21}$$

The tangent to the curve $\vec{g}(t)$ at the point F is given by

$$\vec{T} = (g_1)_t\hat{\imath} + (g_2)_t\hat{\jmath} + (g_3)_t\hat{k} \tag{6.22}$$

The normal vector of the hand, $\vec{\mathbf{n}}$ is assigned equal to $\vec{T}$. The following procedure is adopted to assign the approach vector $\vec{\mathbf{a}}$.

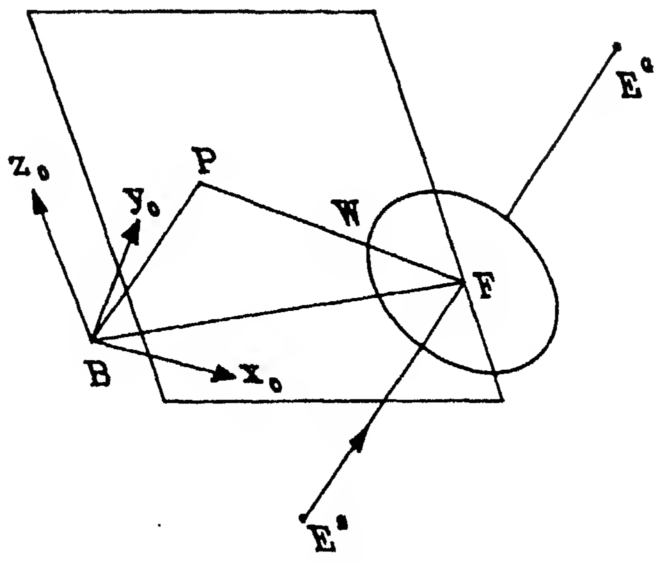


Figure 6.47: Determining the base approach vector

First a plane which is normal to the path $\vec{g}(t)$ at the point F is found. The equation of this plane is given by

$$(g_1)_t(x - x_F) + (g_2)_t(y - y_F) + (g_3)_t(z - z_F) = 0 \tag{6.23}$$

where $(x_F, y_F, z_F)$ are the coordinates of point F. A perpendicular is now dropped from the base to this plane. Let the perpendicular meet the plane at point P (see Figure 6.47). Then the parametric equation of line PF is given by

$$\frac{x - x_B}{(g_1)_t} + \frac{y - y_B}{(g_2)_t} + \frac{z - z_B}{(g_3)_t} = u \text{ (say)} \tag{6.24}$$

where $(x_B, y_B, z_B)$ are the coordinates of the base B. At the point P, the above equation satisfies

$$\begin{aligned} x_P &= (g_1)_t u + x_B \\ y_P &= (g_2)_t u + y_B \\ z_P &= (g_3)_t u + z_B \end{aligned} \tag{6.25}$$

Since point P also lies on the plane defined by (6.23), the following equation also holds.

$$(g_1)_t[(g_1)_t u + x_B)] + (g_2)_t[(g_2)_t u + y_B] + (g_3)_t[(g_3)_t u + z_B] = (g_1)_t x_F + (g_2)_t y_F + (g_3)_t z_F \tag{6.26}$$

which gives rise to the parameter $u$ for point P.

$$u_P = \frac{(g_1)_t(x_F - x_B) + (g_2)_t(y_F - y_B) + (g_3)_t(z_F - z_B)}{(g_1)_t^2 + (g_2)_t^2 + (g_3)_t^2} \tag{6.27}$$

The coordinates of P, $(x_P, y_P, z_P)$ are found by substituting the value of $u_P$ into equations (6.25).

The position of the wrist W is found by positioning W on line FP such that $\overline{\mathrm{WF}} = d_6$. The approach vector, $\vec{a}$ is assigned equal to the vector $\overline{\mathrm{WF}}$. The sliding vector $\vec{s}$ if found from the cross product of $\vec{a}$ and $\vec{n}$.

$$\vec{s} = \vec{a} \times \vec{n}$$

The definition of `class Path` is given in Section C.1. The above method for determining an approach vector is implemented in function `BaseApproachPoint()`. Function `VerticalApproachPoint()` finds the approach vector such that the hand approach vector, $\vec{a}$ is always vertically pointing down.

## Obstacle

An obstacle is a solid cuboid that moves along an assigned path. An obstacle is bounded by `Planes` and is described by assigning its corner vertices. Figure 6.48 shows a case of obstacle representation for a cuboid of length $l_1$, width $l_2$ and height $l_3$. Its vertices are numbered from 1, . . . , 8 and the vertices which decide its bounding planes are 1562, 1485, 2673, 3784, 1234 and 5876 numbered according to the convention described in Section 6.6.1. The description of `class Obstacle` appears in Section C.2.

## Sensor

Sensors are mounted on links to detect obstacles. Obstacles in this case may be of type `class Obstacle` or of type `class Linktype` (to be described later). Sensors

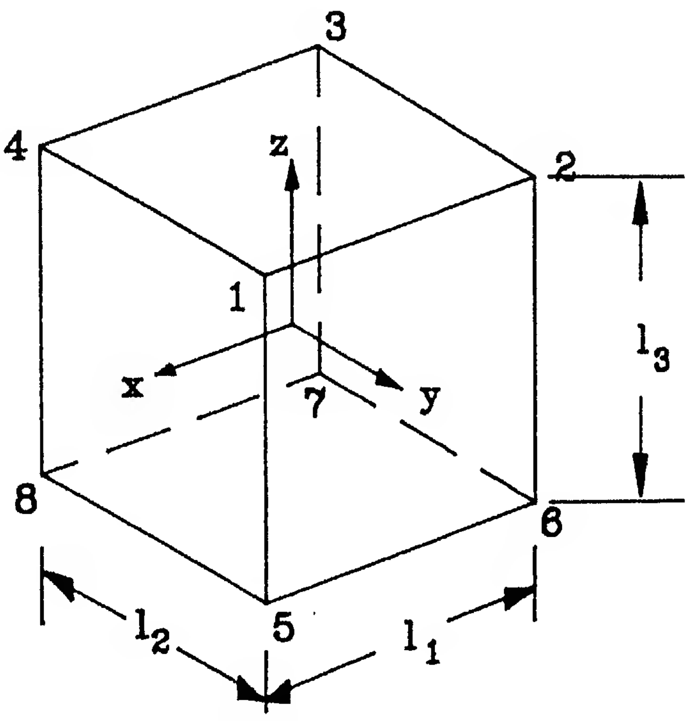


Figure 6.48: Representing an obstacle as a cuboid

will detect a body when the beam of rays intercepts a plane of the obstacle. Each plane of the obstacle is checked for interception, and those that fall within its beam angle, are considered to be candidates for collision.

A brief discussion is presented below to describe the procedure involved for checking interception of a sensor beam with a plane. However, this is only a simplified model and is given only for illustration purpose. More elaborate models which take into account the physical characteristics of sensors are given in [73,72, 71].

In our simplified model, we have idealized the sensor to be a light source. If the beam emitted from a sensor hits an obstacle, the obstacle is assumed to be detected. It is further assumed that sensors will not be able to detect anything beyond a beam angle of $\pm 30°$. The first condition that is imposed for detecting interception is that a ray will intercept a plane only when its normal vector and the ray are in opposite directions. Let $C \equiv (x_C, y_C, z_C)$ be a point which emits a ray (See Figure 6.49). The ray $\vec{c}$ will intersect the plane only if

$$\vec{c} \cdot \vec{m} < 0 \tag{6.28}$$

holds true, i.e. the angle between the vectors $\vec{m}$ and $\vec{c}$ must be greater than 90°.

If the first test is passed, the second test checks whether there exists a point on a plane which lies within the beam angle. Consider Figure 6.49 which shows a ray $\vec{c}$ from the sensor hitting a plane at point T. The normal to the plane is

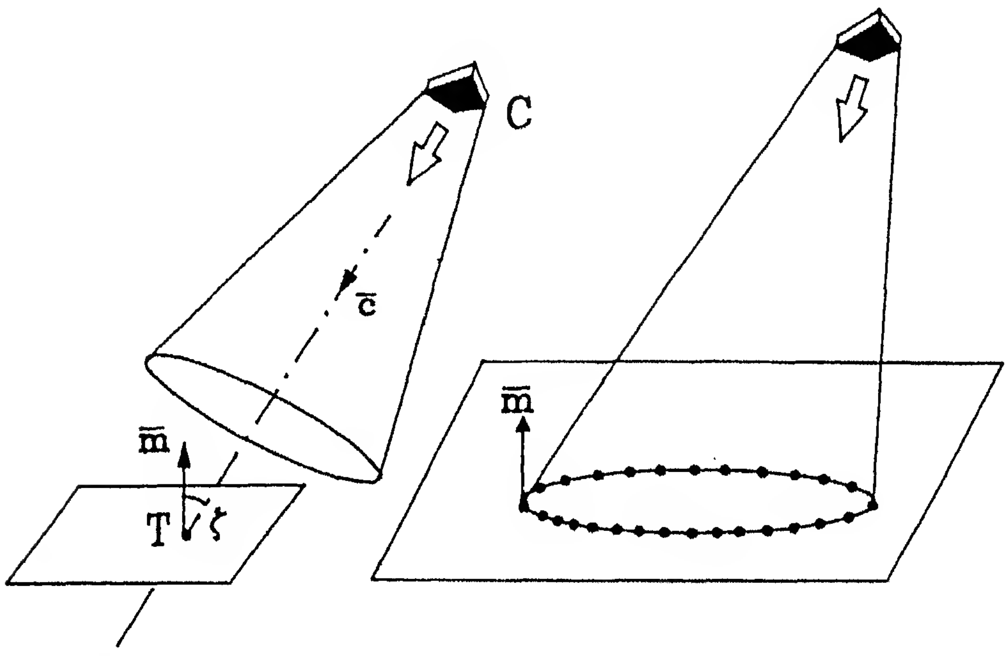


Figure 6.49: Finding the angle between a normal to a plane and a sensor beam

represented by $\vec{m}$. If the angle $\zeta$ between the ray and the normal $\vec{m}$ is less than the beam angle of the sensor, then the point is visible. i.e.

$$\zeta = \cos^{-1} \frac{\vec{m} \cdot \vec{c}}{|\vec{m}|\,|\vec{c}|} < \text{beamangle} \tag{6.29}$$

must hold for the point $T$ to be detectable. Points at various locations, typically the corner points and the centre point, are checked for interception and if any point gives a positive check, the plane is considered to be visible.

If the above does not yield a positive check, a third check is done to ensure that no obstacles are missed. This check is necessary when the plane is too large compared to the size of the beam. The outer periphery of the beam (which is a circle) is considered and its projection on the plane is found. The projection is generally an ellipse. Points are now considered on the periphery of the ellipse and if any point is found to lie within the boundary of the plane, it is concluded that the plane lies within the detectable range of the sensor. This method uses the polygon inclusion test to check for detectability. The description for `class Sensor` appears in Section C.3.

## 6.6.3 Objects for modelling the robot

The object `Robot` is formed from two basic objects — the links and the grasp body. Therefore, this section describes the following objects.

1. links
2. grasp body
3. robot

### Links

Each link has associated with it a transformation matrix (i.e. `class Transformation`) and a set of sensors (i.e. `class Sensor`) at chosen locations. The link parameters described in Tables A.1 and A.2 form part of the data for each link. The description of `class LinkType` is given in Section D.1.

### Grasp body

The grasp body is a special kind of link with a few parameters absent. If the gripper is holding the body, the grasp body becomes a part of the robot and its link transformation is appended to that of the gripper. This link transformation is given in Table 6.2. If the gripper is not holding the grasp body, the grasp body's transformation is its description in the base coordinate frame. It is because of this special property that the grasp object is considered to be a part of the robot, although strictly speaking it could have more appropriately been considered a part of the environment. The description of `class Graspbody` appears in Section D.2.

### Robot

The robot essentially consists of several links, and sometimes may include the grasp body. The data part includes the positions of the *pivot points* and the escape vectors acting there. The description of `class Robot` is given in Section D.3. The important functions of the `class Robot` are used in solving the direct kinematic problem, the inverse kinematic problem and the inverse dynamic problem. The `class Robot` also keeps track of the position of the grasp body and the status of the gripper. It also checks that the kinematic and dynamic constraints are not violated while imparting motion to the joints. The solution strategies adopted when the sensors indicate an impending collision are described in Section 6.4.1.

## 6.6.4 Objects for graphic display

Objects for graphic display are needed for visual observation. The necessary routines are those that enable projecting a three dimensional body on a two dimensional plane. Two kinds of projections are discussed here – parallel projection and perspective projection. Details of other graphic routines like setting up window, viewport and related routines may be obtained from any standard book on computer graphics (See for example [53]).

The primary objects for graphics display are those that first set up the viewing parameters which are then inherited by the window. Thus, the necessary objects are listed as

1. viewing parameters
2. window

### Viewing parameters

The viewing parameters require two projection techniques which are needed for visualizing a three dimensional object on a two-dimensional screen. These are described below.

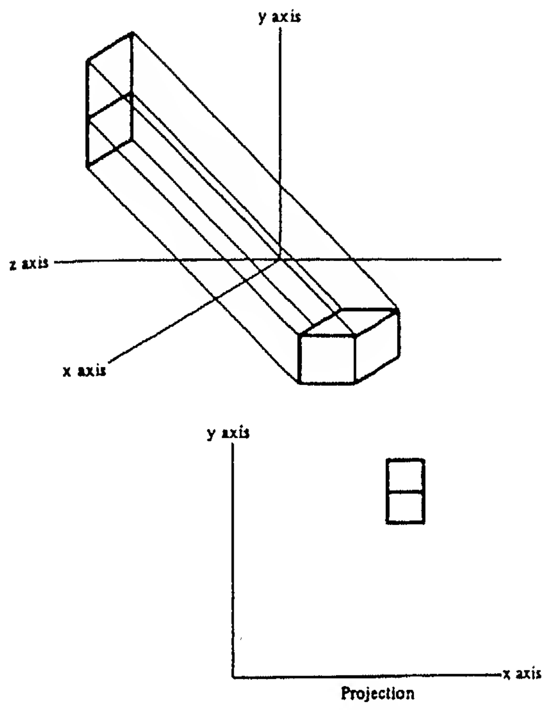


Figure 6.50: Parallel projection

*Parallel Projection :* A parallel projection is formed by extending parallel lines from each vertex on the body until they intersect the plane of the screen. The point of intersection is the projection of the vertex. The projected vertices are connected by line segments which correspond to edges on the original object (See Figure 6.50). Suppose there is a point on the body at $R \equiv (x_R, y_R, z_R)$ and it is desired to determine where the projected point $(x_{R_p}, y_{R_p})$ will lie. Suppose the direction of projection is given by the vector

$$\vec{g} = g_x\hat{\imath} + g_y\hat{\jmath} + g_z\hat{k}$$

and that the image is to be projected onto the $xy$ plane. The parametric equation for a line passing through the point $(x_R, y_R, z_R)$ and in the direction of projection is represented as

$$x = x_R + g_x t$$

$$\begin{aligned} y &= y_R + g_y t \\ z &= z_R + g_z t \end{aligned} \tag{6.30}$$

On the $xy$ plane, $z = 0$ and so

$$t = -\frac{z_R}{g_z} \tag{6.31}$$

Substituting this into the first equation gives

$$\begin{aligned} x_{R_p} &= x_R - z_R(g_x/g_z) \\ y_{R_p} &= y_R - z_R(g_y/g_z) \end{aligned} \tag{6.32}$$

In homogeneous coordinates, (6.32) can be written as

$$\begin{bmatrix} x_{R_p} \\ y_{R_p} \\ z_{R_p} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -g_x/g_z & 0 \\ 0 & 1 & -g_y/g_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_R \\ y_R \\ z_R \\ 1 \end{bmatrix} \tag{6.33}$$

*Perspective Projection :* In perspective projection, the rays are not parallel; instead they converge at single point called the *centre of projection* (see Figure 6.51). It is the intersection of these converging rays with the plane of the screen that determine the projected image. If the centre of projection is $C \equiv (x_C, y_C, z_C)$ and the point on the body is $R \equiv (x_R, y_R, z_R)$, then the projection ray will be the line containing these points and is given by

$$\begin{aligned} x &= x_C + (x_R - x_C)t \\ y &= y_C + (y_R - y_C)t \\ z &= z_C + (z_R - z_C)t \end{aligned} \tag{6.34}$$

The projected point $(x_{R_p}, y_{R_p})$ will be the point where this line intersects the $xy$ plane for which $z = 0$. Thus

$$t = -\frac{z_C}{z_R - z_C} \tag{6.35}$$

Substituting into the first two equations gives

$$\begin{aligned} x_{R_p} &= x_C - z_C\frac{x_R - x_C}{z_R - z_C} \\ y_{R_p} &= y_C - z_C\frac{y_R - y_C}{z_R - z_C} \end{aligned} \tag{6.36}$$

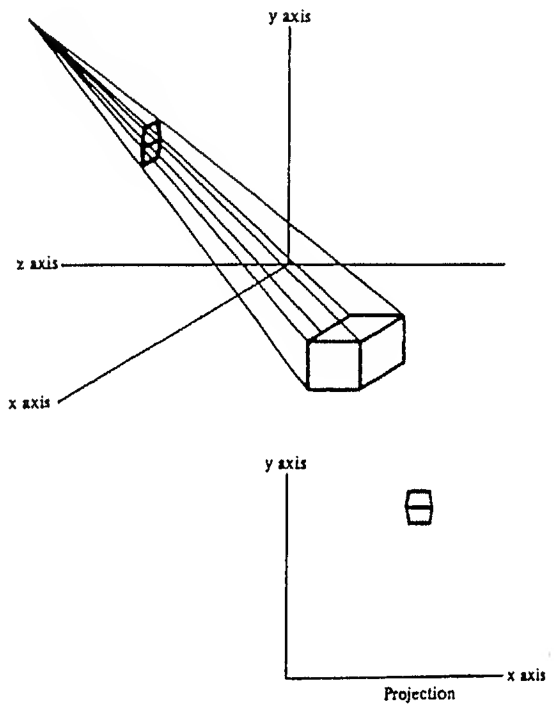


Figure 6.51: Perspective projection

In homogeneous coordinates (6.36) may be written as

$$\begin{bmatrix} x_{R_p} \\ y_{R_p} \\ z_{R_p} \\ 1 \end{bmatrix} = \begin{bmatrix} -z_C & 0 & x_C & 0 \\ 0 & -z_C & y_C & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -z_C \end{bmatrix} \begin{bmatrix} w_R x_R \\ w_R y_R \\ w_R z_R \\ w_R \end{bmatrix} \tag{6.37}$$

where $w_R$ is the ratio $1/(z_R - z_C)$.

The description of `class ViewParameters` is given in Section E.1 which implements the projection techniques described above. For projecting an object on any *arbitrary* plane, first the object is rotated along with the projection plane, so that the plane of projection coincides with the $xy$ plane. The method described above may now be used to find the projected image on the rotated plane.

## Window

The description of `class Window` which implements the drawing routines is given in Section E.2. The drawing and plotting routines are specific to the machine on which the program is implemented.

*Life is the art of drawing sufficient conclusions from insufficient premises.*
– SAMUEL BUTLER

Chapter 7

# Discussions and Concluding Remarks

## 7.1 General Discussions

This thesis used fuzzy logic to develop a fast, simple and flexible algorithm to solve collision avoidance problems for navigation amd manipulation. The applicability of the algorithm has been demonstrated by considering problems in Chapters 4, 5 and 6. Some issues that are of general interest are discussed in this section.

- The choice of bandwidths of angle and distance is crucial for setting up a fuzzy controller. It was found that the bandwidth of angle must be greater than or equal to at least the minimum separation between consecutive elements of the Universe of Angle. The gradation of Universe of Distance was chosen so that it became finer at close distances. This was done to make the algorithm more reliable when measured distances are small, that is when collision is more imminent. The bandwidth of distance must be decided by the minimum separation between the elements appearing near that point along the universe where approaching bodies are presumed to pose a risk of collision.

- It was observed that "a margin of safety" was necessary when working with fuzzy logic. This margin of safety may be reduced if the gradation of the universe (of both angle and distance) is fine. But since it consumes high computer memory it was practical to work with a coarse gradation and incorporate a factor of safety. The factor of safety was incorporated in many ways. The first method was to increase the size of the object in simulation runs. The second method, which is useful for implementations on a real vehicle, was to make the program safe by making the rules trigger somewhat earlier than necessary as an anticipated measure. This was achieved by overdefining the boundary of the term "Near". The third way of incorporating a safety measure was to work with the predicted position of the body rather than the present position, as explained in Section 4.4.2.

- A choice of eight directions for framing the fuzzy rules was guided by the following considerations. It is intuitively easier to give symbolic labels to the eight directions since they are commonly used in navigation and so the framing of rules with eight directions is easy to comprehend. It was observed that using more than eight directions did not lead to significant improvement in performance while requiring more computer memory. On the other hand, taking the number of directions to be less than eight does reduce memory requirements but leads to loss of accuracy.

- The number of sensors needed for the manipulation problems was quite large as compared to that for the navigation problem. The simulation of sensors was done through an approximate but simple approach. The emphasis was upon showing the applicability and effectiveness of the collision avoidance algorithm rather than dealing with the sensors in detail. In practice, a sensor with wider range could be used to reduce the number of sensors. In general, more accurate sensors are needed for manipulation problems compared to navigation problems, because the workspace of a manipulator has less margin of safety than that for a mobile vehicle.

- Typical execution time found during simulation runs for navigation problems on an IBM-AT when working with two rules and two bodies, was of the order of half a second for each step. For five bodies and three rules the execution time was 3.3 seconds. When working with a planar rotary four-link manipulator the execution time was found to be 2.5 seconds for each step. For simulation of the PUMA-560 using fuzzy rules, the execution time for each step was found to be 3.4 seconds. These figures indicate that the algorithm is reasonably fast and may be used for real-time computation on small machines.

- The plots of the joint torque for manipulators was found to be irregular. This is so because a local planning strategy based on instantaneous data was used. A planning strategy which considers a global map can produce smooth torque graphs.

- The fuzzy-rule based methodology for collision avoidance of manipulator arms is somewhat similar to the artificial potential field approach [65]. However, it has a few differences which are pointed out below:

  1. The setting up of the potential functions (FIRAS) and finding an avoidance vector will always give the direction of approach exactly opposite to the direction of obstacle. The method proposed in this work however is not limited to yield only one direction, since it may be 'tuned' to avoid the obstacle in any favourable direction. This is decided mainly by the structure of the manipulator. For example, avoidance measures might require that the entire manipulator retract towards its base when faced with an obstacle near its terminal links instead of moving away directly opposite to it. Inclusion of such avoidance measures are easy to incorporate by changing parameters of the rule-base governing the escape strategies.
  2. The obstacle shapes are not modeled explicitly (as super-quadrics). The proximity of an obstacle from a link is assessed through sensors mounted on the surface of links. The proximity information is converted to a symbolic representation (by fuzzyfying the input) and a fuzzy controller

advises how much the links must move to avoid an impending collision. This gives the best desired goal state under the present circumstances. The actual actuator motions are calculated at a later stage by a geometric planner which considers the motion constraints like link connectivity, joint limits, joint velocities, joint accelerations and joint torques. It is thus attempted to decouple the goal states from actual actuator motion as suggested in [84].

3. A totally unknown environment has been assumed prior to start of execution. Thus the initial motion of the manipulator is decided by the shortest route to the goal point.

## 7.2 Concluding Remarks

The conclusions drawn from the thesis are presented in the following paragraphs.

1. Fuzzy logic was shown to be useful for approximate and qualitative motion planning in an unknown environment for both navigation and manipulation problems.

2. It was found that qualitative avoidance measures were adequate for most practical planning problems in uncluttered environments. Minor inaccuracies in sensory data were smoothened out by fuzzyfying the input. (These inaccuracies were artificially introduced in simulation runs.)

3. The proposed approach was shown to be fairly fast because of the following reasons:

   (a) The calculations were found in the cartesian space using sensory data thus avoiding the computationally expensive configuration space.

   (b) The algorithm needed information about only the vicinity of concerned objects rather than a complete global map.

   (c) The fuzzy algorithm was used to decide the best possible goal state for the next instant under the present circumstances. Then a geometric planner was used to achieve the suggested goal state as closely as possible without violating the kinematic constraints. This avoided re-calculation of the goal state, whereas in the earlier works the goal state and the geometric planner were inseparable.

4. The fuzzy rules were shown to be flexible in the sense that these rules could be modified easily to accommodate the changes in the requirements (See Section 5.5.3).

5. The assignment of meaning to a term was found to depend on the amount of safety desired. It was demonstrated in Section 4.4.6 that most of the control parameters used in the fuzzy algorithm need to be *above* a certain optimum level for best performance. It was also observed that assigning a very high value did not lead to any further improvement. The correct value for a parameter was best found through simulation runs.

6. It was found that fuzzy logic is not very useful for accurate motion planning (e.g., when the object needs to graze past obstacles,) since an accurate description will require much finer gradation of the universe which in turn needs large computer memory.

7. The proposed method of navigation in this work is suitable only for local navigation since it is based on locally perceived instantaneous data. The suggested navigation strategy will therefore fail in "maze solving" problems.

## 7.3 Scope for Future Work

A suggested extension of this work is to implement the algorithm on a real autonomous vehicle and a robot manipulator. For guiding the choice of sensors, it is suggested that inexpensive sensors with less accuracy be used in large quantities than fewer number of accurate sensors.

The navigation problem solved in Chapter 4 may be extended to include elongated bodies (like a car) with non-holonomic constraints on its motion. A further extension is in solving the 3-D navigation problem (commonly called the Asteroid

Avoidance Problem). The inclusion of "Speed" as an explicit variable in the fuzzy rules is also a suggested possibility.

For manipulation, it is suggested that the proposed planning strategy be coupled with another more accurate planning strategy working side by side. Since the proposed method is faster but less accurate, it may be used at the gross level. When fine motion planning is required, the other strategy being more accurate, might be more appropriate.

Appendix A

# Description of PUMA-560 Robot

Modelling a robot for simulation requires a complete analysis of its kinematic and dynamic characteristics. In this appendix the link coordinate systems are first established, transformation matrices are formulated, and direct kinematics problem is solved.

First, the coordinate systems for each link are established. These are required for further analysis of the robot.

## A.1 Link Coordinate Systems

A detailed analysis of the material covered in this section can be obtained from [44]. Figure A.1 shows the location of the orthonormal coordinate systems attached to the origin of each link. The Denavit-Hartenberg (D-H) $4 \times 4$ transformation matrix is used to represent the each link's coordinate system at the joint with the previous link's coordinate system. The coordinate frame $(x_0, y_0, z_0)$ denotes the base coordinate system. The D-H representation of a rigid link depends on four geometric parameters associated with each link. These four parameters completely describe any revolute or prismatic joint. (The PUMA-560 incidentally has only revolute joints). Referring to Figure A.1, these parameters are defined as follows:

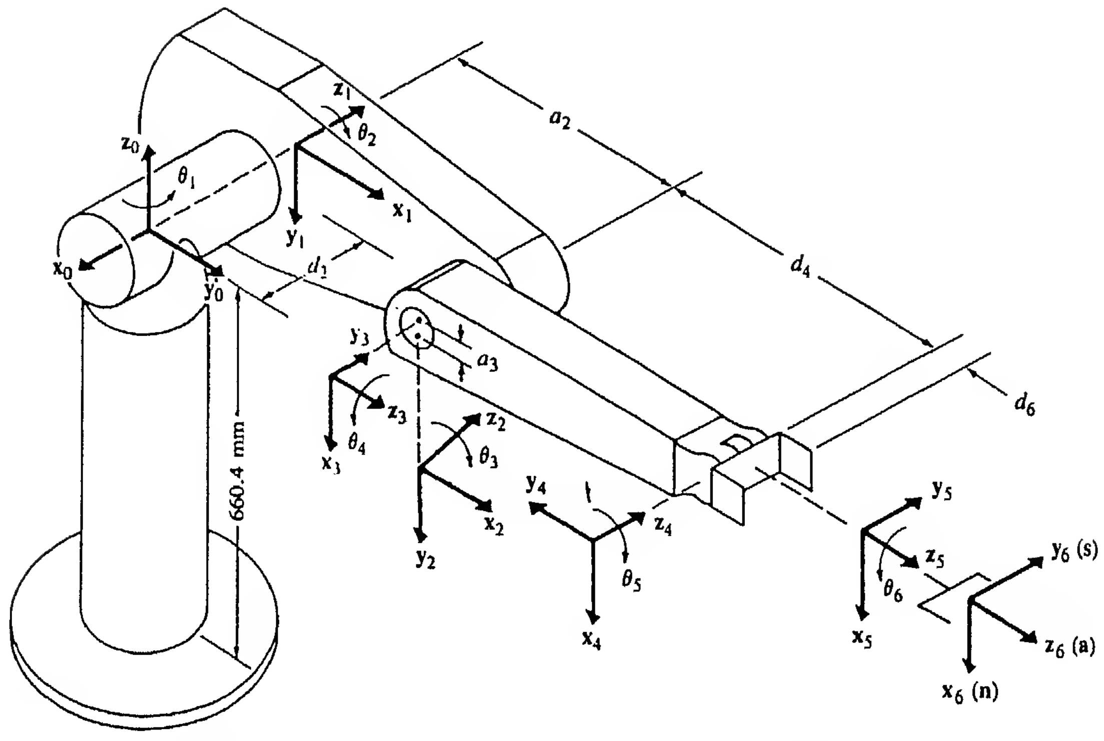


Figure A.1: The link coordinate system for the PUMA-560 robot

$\theta_i$ is the joint angle from $\mathbf{x}_{i-1}$ axis to the $\mathbf{x}_i$ axis about the $\mathbf{z}_{i-1}$ axis (using the right hand rule).

$d_i$ is the distance from the origin of the $(i-1)^{th}$ coordinate frame to the intersection of the $\mathbf{z}_{i-1}$ with the $\mathbf{x}_i$ axis along the $\mathbf{z}_{i-1}$ axis.

$a_i$ is the offset distance from the intersection of the $\mathbf{z}_{i-1}$ axis with the $\mathbf{x}_i$ axis to the origin of the $i^{th}$ frame along the $\mathbf{x}_i$ (or the shortest distance between the $\mathbf{z}_{i-1}$ and $\mathbf{z}_i$ axes).

$\alpha_i$ is the offset angle from the $\mathbf{z}_{i-1}$ axis to the $\mathbf{z}_i$ axis about the $\mathbf{x}_i$ axis (using the right-hand rule).

Accordingly the link coordinate parameters of the PUMA-560 robot are given in Table A.1. For a rotary joint, $d_i$, $a_i$ and $\alpha_i$ are the link parameters and remain constant for a robot, while $\theta_i$ is the joint variable that changes when link $i$ rotates with respect to link $i-1$.

| PUMA robot arm link coordinate parameters | | | | | |
|---|---|---|---|---|---|
| *Joint i* | $\theta_i$ | $\alpha_i$ | $a_i$ | $d_i$ | *Joint range* (degrees) |
| 1 | $\theta_1 + 90$ | −90 | 0 | 0 | −160 to +160 |
| 2 | $\theta_2$ | 0 | 431.8 mm | 149.09 mm | −225 to +45 |
| 3 | $\theta_3 + 90$ | 90 | −20.32 mm | 0 | −45 to +225 |
| 4 | $\theta_4$ | −90 | 0 | 433.07 mm | −110 to +170 |
| 5 | $\theta_5$ | 90 | 0 | 0 | −100 to +100 |
| 6 | $\theta_6$ | 0 | 0 | 56.25 mm | −266 to +266 |

Table A.1: Link parameters for the PUMA-560 robot

Once the D-H coordinate system has been established for each link, a homogeneous transformation relating the $i^{th}$ coordinate frame to the $(i-1)^{th}$ coordinate frame is formed. The composite homogeneous transformation matrix ${}^{i-1}\mathbf{A}_i$ known as the D-H transformation matrix for adjacent coordinate frames $i$ and $i-1$ is expressed by (A.1).

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

The inverse of this matrix which relates the $(i-1)^{th}$ coordinate frame to the $i^{th}$ coordinate frame can be found to be

$$ [{}^{i-1}\mathbf{A}_i]^{-1} = {}^{i}\mathbf{A}_{i-1} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 & -a_i \\ -\cos\alpha_i\sin\theta_i & \cos\alpha_i\cos\theta_i & \sin\alpha_i & -d_i\sin\alpha_i \\ \sin\alpha_i\sin\theta_i & -\sin\alpha_i\cos\theta_i & \cos\alpha_i & -d_i\cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

The inverse matrix is required when solving the inverse dynamics problem.

If we introduce the short form notation $C_i \equiv \cos\theta_i$, $S_i \equiv \sin\theta_i$, $C_{ij} \equiv \cos(\theta_i + \theta_j)$ and $S_{ij} \equiv \sin(\theta_i + \theta_j)$ then the transformation matrices for each link of the PUMA-560 are as follows.

$$ {}^{0}\mathbf{A}_1 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

$$ {}^{1}\mathbf{A}_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.4} $$

$$ {}^{2}\mathbf{A}_3 = \begin{bmatrix} C_3 & 0 & S_3 & a_3C_3 \\ S_3 & 0 & -C_3 & a_3S_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.5} $$

$$ {}^{3}\mathbf{A}_4 = \begin{bmatrix} C_4 & 0 & -S_4 & 0 \\ S_4 & 0 & C_4 & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.6} $$

$$ {}^{4}\mathbf{A}_5 = \begin{bmatrix} C_5 & 0 & S_5 & 0 \\ S_5 & 0 & -C_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.7} $$

$$ {}^{5}\mathbf{A}_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.8} $$

The position and orientation of the end-point of the manipulator with respect to the base coordinate system is represented by the transformation matrix $\mathbf{T} = {}^{0}\mathbf{A}_6$ and is frequently referred as the "arm matrix". The arm matrix can be expressed in the form

$$ \begin{aligned} \mathbf{T} &= \begin{bmatrix} \vec{\mathrm{n}} & \vec{\mathrm{s}} & \vec{\mathrm{a}} & \vec{\mathbf{p}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{A.9} $$

where (see Figure A.2)

$\vec{\mathrm{n}} =$ normal vector of the hand. Assuming a parallel-jaw hand, it is orthogonal to the fingers of the robot arm.

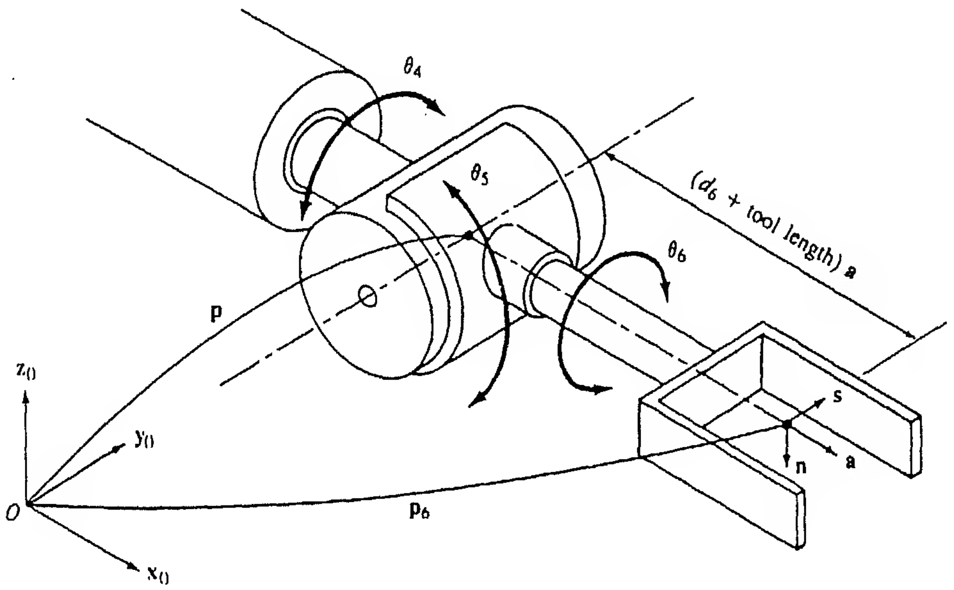


Figure A.2: Hand coordinate system and $[\vec{n}, \vec{s}, \vec{a}]$.

$\vec{s}$ = sliding vector of the hand. It is pointing in the direction of the finger motion as the gripper opens and closes.

$\vec{a}$ = approach vector of the hand. It is pointing in the direction normal to the palm of the hand (i.e. normal to the tool mounting plate of the arm).

$\vec{p}$ = position vector of the hand. It points from the origin of the base coordinate system to the origin of the hand coordinate system, which is usually located at the centre point of the fully closed fingers.

## A.2 The Direct Kinematics Problem

The direct kinematic solution addresses the problem of finding the end-effector location with all joint angles known. This is a simple problem because it involves finding out the arm matrix **T** for the given set of joint values. The direct kinematics problem yields the position of the end-effector $\vec{p}_6$ and the normal, approach and sliding vectors $[\vec{n}, \vec{s}, \vec{a}]$. The components of the arm matrix of (A.9) can be found out by matrix multiplication.

$$n_x = C_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5C_6] - S_1(S_4C_5C_6 + C_4S_6)$$

$$\begin{aligned}
n_y &= S_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5C_6] + C_1(S_4C_5C_6 + C_4S_6) \\
n_z &= -S_{23}(C_4C_5C_6 - S_4S_6) - C_{23}S_5C_6 && \text{(A.10)} \\
s_x &= C_1[-C_{23}(C_4C_5S_6 + S_4C_6) + S_{23}S_5S_6] - S_1(-S_4C_5S_6 + C_4C_6) \\
s_y &= S_1[-C_{23}(C_4C_5S_6 + S_4C_6) + S_{23}S_5S_6] + C_1(-S_4C_5S_6 + C_4C_6) \\
s_z &= S_{23}(C_4C_5S_6 + S_4C_6) + C_{23}S_5S_6 && \text{(A.11)} \\
a_x &= C_1(C_{23}C_4S_5 + S_{23}C_5) - S_1S_4S_5 \\
a_y &= S_1(C_{23}C_4S_5 + S_{23}C_5) + C_1S_4S_5 \\
a_z &= -S_{23}C_4S_5 + C_{23}C_5 && \text{(A.12)} \\
p_x &= C_1[d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} + a_2C_2] - S_1(d_6S_4S_5 + D_2) \\
p_y &= S_1[d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} + a_2C_2] + C_1(d_6S_4S_5 + D_2) \\
p_z &= d_6(C_{23}C_5 - S_{23}C_4S_5) + C_{23}d_4 - a_3S_{23} - a_2S_2 && \text{(A.13)}
\end{aligned}$$

For simulation, it is also necessary to compute the intermediate matrices ${}^0\mathbf{A}_i$ for $i = 2, \ldots, n-1$. The intermediate matrices for $i = 2, 3, 4$ and 5 are reproduced below.

$$ {}^0\mathbf{A}_2 = \begin{bmatrix} C_1C_2 & -C_1S_2 & -S_1 & a_2C_1C_2 - d_2S_1 \\ S_1C_2 & -S_1S_2 & C_1 & a_2S_1C_2 + d_2C_1 \\ -S_2 & -C_2 & 0 & -a_2S_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.14}$$

$$ {}^0\mathbf{A}_3 = \begin{bmatrix} C_1C_{23} & -S_1 & C_1S_{23} & a_2C_1C_2 + a_3C_1C_{23} - d_2S_1 \\ S_1C_{23} & C_1 & S_1S_{23} & a_2S_1C_2 + a_3S_1C_{23} + d_2C_1 \\ -S_{23} & 0 & C_{23} & -a_2S_2 - a_3S_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.15}$$

$$ {}^0\mathbf{A}_4 = \begin{bmatrix} ({}^0n_4)_x & ({}^0s_4)_x & ({}^0a_4)_x & ({}^0p_4)_x \\ ({}^0n_4)_y & ({}^0s_4)_y & ({}^0a_4)_y & ({}^0p_4)_y \\ ({}^0n_4)_z & ({}^0s_4)_z & ({}^0a_4)_z & ({}^0p_4)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.16}$$

where

$$\begin{aligned}
({}^0n_4)_x &= C_1C_{23}C_4 - S_1S_4 \\
({}^0s_4)_x &= -C_1S_{23} \\
({}^0a_4)_x &= -C_1C_{23}S_4 - S_1C_4 \\
({}^0p_4)_x &= d_4C_1S_{23} + a_2C_1C_2 + a_3C_1C_{23} - d_2S_1 \\
({}^0n_4)_y &= S_1C_{23}C_4 + C_1S_4 \\
({}^0s_4)_y &= -S_1S_{23}
\end{aligned}$$

$$\begin{aligned}
({}^0a_4)_y &= -S_1C_{23}S_4 + C_1C_4 \\
({}^0p_4)_y &= d_4S_1S_{23} + a_2S_1C_2 + a_3S_1C_{23} + d_2C_1 \\
({}^0n_4)_z &= -S_{23}C_4 \\
({}^0s_4)_z &= -C_{23} \\
({}^0a_4)_z &= S_{23}S_4 \\
({}^0p_4)_z &= d_4C_{23} - a_2S_2 - a_3S_{23}
\end{aligned}$$

$$ {}^0\mathbf{A}_5 = \begin{bmatrix} ({}^0n_5)_x & ({}^0s_5)_x & ({}^0a_5)_x & ({}^0p_5)_x \\ ({}^0n_5)_y & ({}^0s_5)_y & ({}^0a_5)_y & ({}^0p_5)_y \\ ({}^0n_5)_y & ({}^0s_5)_y & ({}^0a_5)_y & ({}^0p_5)_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.17}$$

where

$$\begin{aligned}
({}^0n_5)_x &= C_1C_{23}C_4C_5 - S_1S_4C_5 - C_1S_{23}S_5 \\
({}^0s_5)_x &= -C_1C_{23}S_4 - S_1C_4 \\
({}^0a_5)_x &= C_1C_{23}C_4S_5 - S_1S_4S_5 + C_1S_{23}C_5 \\
({}^0p_5)_x &= d_4C_1S_{23} + a_2C_1C_2 + a_3C_1C_{23} - d_2S_1 \\
({}^0n_5)_y &= S_1C_{23}C_4C_5 + C_1S_4C_5 - S_1S_{23}S_5 \\
({}^0s_5)_y &= -S_1C_{23}S_4 + C_1C_4 \\
({}^0a_5)_y &= S_1C_{23}C_4S_5 + C_1S_4S_5 + S_1S_{23}C_5 \\
({}^0p_5)_y &= d_4S_1S_{23} + a_2S_1C_2 + a_3S_1C_{23} + d_2C_1 \\
({}^0n_5)_z &= -S_{23}C_4C_5 - C_{23}S_5 \\
({}^0s_5)_z &= S_{23}S_4 \\
({}^0a_5)_z &= C_{23}C_5 - S_{23}C_4S_5 \\
({}^0p_5)_z &= d_4C_{23} - a_2S_2 - a_3S_{23}
\end{aligned}$$

## A.3 Geometric Solution for Inverse Kinematics

In this section a geometric solution strategy is presented for the inverse kinematic problem. The solution strategy is unique for the PUMA-560 and is not general. However, solutions for other robot manipulators may be attempted on similar lines, but this issue is kept aside because it is beyond the interest of this thesis. Various geometric solution strategies for the PUMA have been proposed (see [35,36] for example) but the one referred in [44] is chosen here.

For the PUMA robot various arm configurations are defined according to human arm geometry. These are specified by the user for finding the inverse kinematic solution.

$$\text{ARM} = \begin{cases} +1 & \text{RIGHT arm} \\ -1 & \text{LEFT arm} \end{cases} \tag{A.18}$$

$$\text{ELBOW} = \begin{cases} +1 & \text{ABOVE arm} \\ -1 & \text{BELOW arm} \end{cases} \tag{A.19}$$

$$\text{WRIST} = \begin{cases} +1 & \text{WRIST DOWN} \\ -1 & \text{WRIST UP} \end{cases} \tag{A.20}$$

$$\text{FLIP} = \begin{cases} +1 & \text{Flip the wrist orientation} \\ -1 & \text{Do not flip the wrist orientation} \end{cases} \tag{A.21}$$

**Arm solution for the first three joints**

From the kinematic diagram of the PUMA robot arm in Figure A.1, a position vector which points from the base B to the position of the wrist W $\equiv (x_W, y_W, z_W)$ is defined.

$$\begin{aligned} {}^0\vec{\mathbf{p}}_4 &= [({}^0p_4)_x, [({}^0p_4)_y, [({}^0p_4)_z]^T \\ &= [x_W, y_W, z_W]^T \end{aligned}$$

*Joint 1 solution.* With reference to Figure A.3, the solution for $\theta_1$ is given by (for details refer to [44])

$$\theta_1 = \tan^{-1}\left(\frac{-\text{ARM} y_W\sqrt{x_W^2 + y_W^2 - d_2^2} - x_W d_2}{-\text{ARM} x_W\sqrt{x_W^2 + y_W^2 - d_2^2} + y_W d_2}\right) \qquad -\pi \le \theta_1 \le \pi \tag{A.22}$$

*Joint 2 solution.* The solution for $\theta_2$ depends on the configuration parameters ARM and ELBOW. The position vector ${}^0\vec{\mathbf{p}}_4$ is projected on the plane $\mathbf{x}_1\mathbf{y}_1$ and the following parameters are obtained from the arm geometry (see Figure A.4)

$$R = \sqrt{x_W^2 + y_W^2 + z_W^2 - d_2^2} \qquad r = \sqrt{x_W^2 + y_W^2 - d_2^2} \tag{A.23}$$

$$\sin\alpha = -\frac{z_W}{R} = -\frac{z_W}{\sqrt{x_W^2 + y_W^2 + z_W^2 - d_2^2}} \tag{A.24}$$

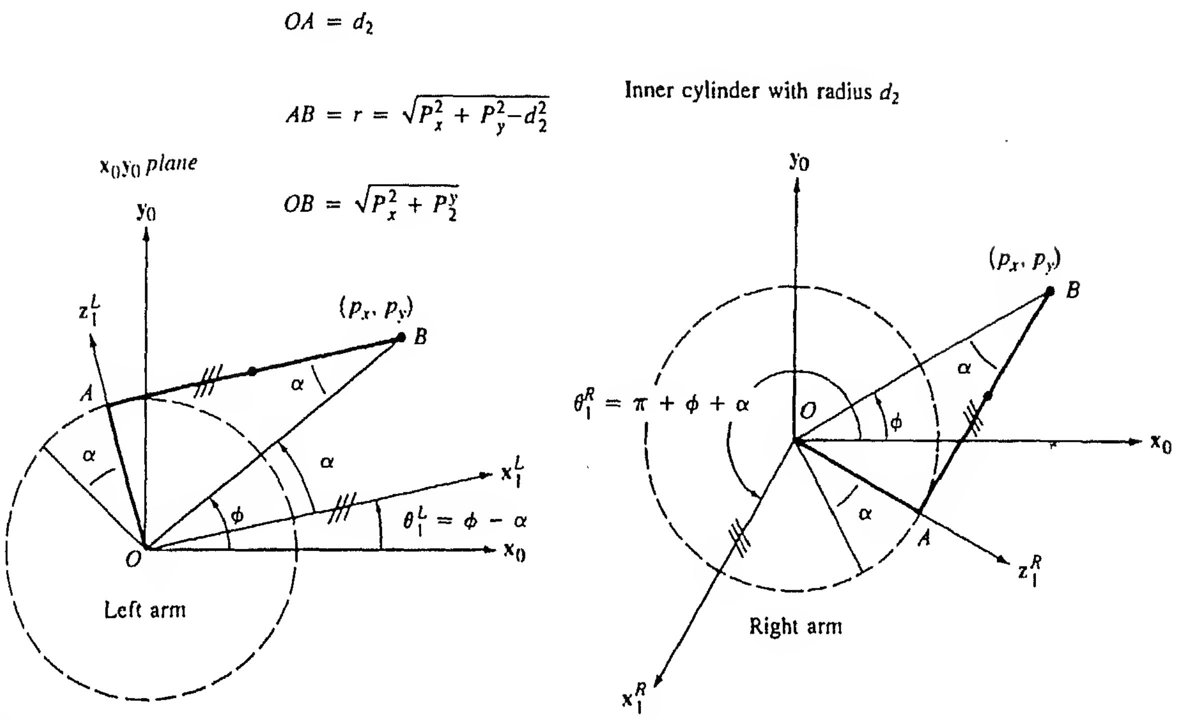

$OA = d_2$
$AB = r = \sqrt{P_x^2 + P_y^2 - d_2^2}$
$OB = \sqrt{P_x^2 + P_y^2}$
$x_0y_0$ plane
$y_0$
$z_1^L$
$(p_x, p_y)$
B
A
$\alpha$
O
$\phi$
$x_1^L$
$\theta_1^L = \phi - \alpha$
$x_0$
Left arm
Inner cylinder with radius $d_2$
$\theta_1^R = \pi + \phi + \alpha$
$z_1^R$
$x_1^R$
Right arm

Figure A.3: Solution for joint 1 of PUMA-560

$$\cos\alpha = -\frac{\text{ARM}\cdot r}{R} \tag{A.25}$$

$$= -\frac{\text{ARM}\cdot\sqrt{x_W^2 + y_W^2 - d_2^2}}{\sqrt{x_W^2 + y_W^2 + z_W^2 - d_2^2}} \tag{A.26}$$

$$\cos\beta = \frac{a_2^2 + R^2 - (d_4^2 + a_3^2)}{2a_2R} \tag{A.27}$$

$$= \frac{a_2^2 + x_W^2 + y_W^2 + z_W^2 - d_2^2 - (d_4^2 + a_3^2)}{2a_2\sqrt{x_W^2 + y_W^2 + z_W^2 - d_2^2}} \tag{A.28}$$

$$\sin\beta = \sqrt{1 - \cos^2\beta} \tag{A.29}$$

(The "$\cdot$" operation is the multiplication operation on the indicators). Now $\theta_2$ is found from the expression

$$\theta_2 = \alpha + (\text{ARM}\cdot\text{ELBOW})\beta \tag{A.30}$$

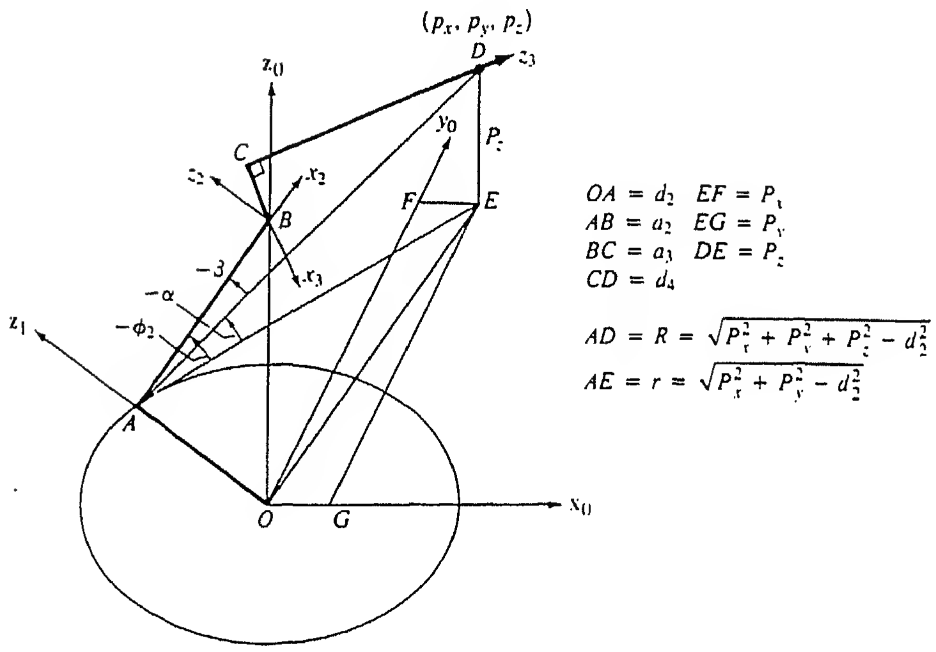

$(p_x, p_y, p_z)$
$D$
$z_3$
$z_0$
$y_0$
$P_z$
$C$
$z_2$
$x_2$
$B$
$F$
$E$
$-\beta$
$x_3$
$-\alpha$
$z_1$
$-\phi_2$
$A$
$O$
$G$
$x_0$
$OA = d_2 \quad EF = P_x$
$AB = a_2 \quad EG = P_y$
$BC = a_3 \quad DE = P_z$
$CD = d_4$
$AD = R = \sqrt{P_x^2 + P_y^2 + P_z^2 - d_2^2}$
$AE = r = \sqrt{P_x^2 + P_y^2 - d_2^2}$

Figure A.4: Solution for joint 2 of PUMA-560

which leads to

$$\sin\theta_2 = \sin\alpha\cos\beta + (\text{ARM}\cdot\text{ELBOW})\cos\alpha\sin\beta \tag{A.31}$$

$$\cos\theta_2 = \cos\alpha\cos\beta - (\text{ARM}\cdot\text{ELBOW})\sin\alpha\sin\beta. \tag{A.32}$$

The value of $\theta_2$ is now found from

$$\theta_2 = \tan^{-1}\left(\frac{\sin\theta_2}{\cos\theta_2}\right) \qquad -\pi \le \theta_2 \le \pi \tag{A.33}$$

*Joint 3 solution.* For joint 3, the position vector ${}^0\vec{\mathbf{p}}_4$ is projected onto the $\mathbf{x}_2\mathbf{y}_2$ plane as shown in Figure A.5. According to the figure, the following additional parameters may be defined.

$$\cos\phi = \frac{a_2^2 + (d_4^2 + a_3^2) - R^2}{2a_2\sqrt{d_4^2 + a_3^2}} \tag{A.34}$$

$$\sin\phi = \text{ARM}\cdot\text{ELBOW}\sqrt{1 - \cos^2\phi} \tag{A.35}$$

$$\sin\gamma = \frac{d_4}{\sqrt{d_4^2 + a_3^2}} \tag{A.36}$$

$$\cos\gamma = \frac{|a_3|}{\sqrt{d_4^2 + a_3^2}} \qquad \text{(A.37)}$$

Angle $\theta_3$ can be expressed as

$$\theta_3 = \phi - \gamma \qquad \text{(A.38)}$$

which leads to

$$\sin\theta_3 = \sin\phi\cos\gamma - \cos\phi\sin\gamma \qquad \text{(A.39)}$$
$$\cos\theta_3 = \cos\phi\cos\gamma - \sin\phi\sin\gamma \qquad \text{(A.40)}$$

The value of $\theta_3$ is now found from

$$\theta_3 = \tan^{-1}\left(\frac{\sin\theta_3}{\cos\theta_3}\right) \qquad -\pi \leq \theta_3 \leq \pi \qquad \text{(A.41)}$$

**Solution for the last three joints**

*Joint 4 solution.* An orientation indicator $\Upsilon$ is first defined as follows.

$$\Upsilon = \begin{cases} 0 & \text{if } \mathbf{z_3} \text{ is parallel to } \vec{\mathbf{a}} \\ \vec{\mathbf{s}} \cdot \frac{(\mathbf{z_3} \times \vec{\mathbf{a}})}{\|\mathbf{z_3} \times \vec{\mathbf{a}}\|} & \text{if } \vec{\mathbf{s}} \cdot (\mathbf{z_3} \times \vec{\mathbf{a}}) \neq 0 \\ \vec{\mathbf{n}} \cdot \frac{(\mathbf{z_3} \times \vec{\mathbf{a}})}{\|\mathbf{z_3} \times \vec{\mathbf{a}}\|} & \text{if } \vec{\mathbf{s}} \cdot (\mathbf{z_3} \times \vec{\mathbf{a}}) = 0 \end{cases} \qquad \text{(A.42)}$$

A parameter $M$ is now defined as $M$ = WRIST $sign(\Upsilon)$ where the "*sign*" function is as defined in (4.11). The solution for $\theta_4$ is given by

$$\theta_4 = \tan^{-1}\left(\frac{M\ (C_1 a_y - S_1 a_x)}{M\ (C_1 C_{23} a_x + S_1 C_{23} a_y - S_{23} a_z)}\right) \qquad -\pi \leq \theta_4 \leq \pi \qquad \text{(A.43)}$$

*Joint 5 solution.* To find $\theta_5$ the criterion which aligns the axis of rotation of joint 6 with the approach vector is used, i.e. $\vec{\mathbf{a}} = \mathbf{z_5}$. Taking the projection of the coordinate frame $(\mathbf{x_5}, \mathbf{y_5}, \mathbf{z_5})$ on the $\mathbf{x_4}\mathbf{y_4}$ plane, it can be shown that

$$\sin\theta_5 = \vec{\mathbf{a}} \cdot \mathbf{x_4} \qquad \cos\theta_5 = -(\vec{\mathbf{a}} \cdot \mathbf{y_4}) \qquad \text{(A.44)}$$

The $x$ and $y$ column vectors of ${}^0\mathbf{A}_4$ are now used to deduce

$$\theta_5 = \tan^{-1}\left(\frac{(C_1 C_{23} C_4 - S_1 S_4) a_x + (S_1 C_{23} C_4 + C_1 S_4) a_y - C_4 S_{23} a_z}{C_1 S_{23} a_x + S_1 S_{23} a_y + C_{23} a_z}\right)$$
$$-\pi \leq \theta_5 \leq \pi \qquad \text{(A.45)}$$

*Joint 6 solution.* Projecting the hand coordinate frame $(\vec{n}, \vec{s}, \vec{a})$ on the $\mathbf{x}_5\mathbf{y}_5$ plane, it can be shown that

$$\sin\theta_6 = \vec{n}\cdot\mathbf{y}_5 \qquad \cos\theta_6 = \vec{s}\cdot\mathbf{y}_5 \tag{A.46}$$

where $\mathbf{y}_5$ is the $y$ column vector of ${}^0\mathbf{A}_5$ and $\vec{n}$ and $\vec{s}$ are the normal and sliding vectors of ${}^0\mathbf{A}_6$ , respectively. Thus, the solution for $\theta_6$ is

$$\begin{aligned}\theta_6 &= \tan^{-1}\left(\frac{(-S_1C_4 - C_1C_{23}S_4)n_x + (C_1C_4 - S_1C_{23}S_4)n_y + (S_4S_{23})n_z}{(-S_1C_4 - C_1C_{23}S_4)s_x + (C_1C_4 - S_1C_{23}S_4)s_y + (S_4S_{23})s_z}\right)\\ &\quad -\pi \le \theta_5 \le \pi\end{aligned} \tag{A.47}$$

## A.4 Jacobian for PUMA-560

The derivation for the terms in the Jacobian matrix are presented in this section. For revolute joints the Jacobian matrix is defined as

$$\left\{{}^0\mathrm{J}_k\right\} = \left\{\begin{array}{c} ({}^{k-1}p_6)_x({}^0s_{k-1})_x - ({}^{k-1}p_6)_y({}^0n_{k-1})_x \\ ({}^{k-1}p_6)_x({}^0s_{k-1})_y - ({}^{k-1}p_6)_y({}^0n_{k-1})_y \\ ({}^{k-1}p_6)_x({}^0s_{k-1})_z - ({}^{k-1}p_6)_y({}^0n_{k-1})_z \\ ({}^0a_{k-1})_x \\ ({}^0a_{k-1})_y \\ ({}^0a_{k-1})_z \end{array}\right\} (k = 1,\ldots,6) \tag{A.48}$$

For evaluating the terms ${}^{k-1}\vec{\mathbf{p}}_6$, the matrices ${}^{k-1}\mathbf{A}_6$ are necessary. These matrices for the PUMA-560 are reproduced below for convenience.

$${}^5\mathbf{A}_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.49}$$

$${}^4\mathbf{A}_6 = \begin{bmatrix} C_5C_6 & -C_5S_6 & S_5 & S_5d_6 \\ S_5C_6 & -S_5S_6 & -C_5 & -C_5d_6 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.50}$$

$${}^3\mathbf{A}_6 = \begin{bmatrix} C_4C_5C_6 - S_4S_6 & -C_4C_5S_6 - S_4C_6 & C_4S_5 & C_4S_5d_6 \\ S_4C_5C_6 + C_4S_6 & -S_4C_5S_6 + C_4C_6 & S_4S_5 & S_4S_5d_6 \\ -S_5C_6 & S_5S_6 & C_5 & C_5d_6 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.51}$$

$$^2\mathbf{A}_6 = \begin{bmatrix} (^2n_6)_x & (^2s_6)_x & (^2a_6)_x & (^2p_6)_x \\ (^2n_6)_y & (^2s_6)_y & (^2a_6)_y & (^2p_6)_y \\ (^2n_6)_z & (^2s_6)_z & (^2a_6)_z & (^2p_6)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.52}$$

where

$$\begin{aligned}
(^2n_6)_x &= C_3(C_4C_5C_6 - S_4S_6) - S_3S_5C_6 \\
(^2s_6)_x &= C_3(-C_4C_5S_6 - S_4C_6) + S_3S_5S_6 \\
(^2a_6)_x &= C_3C_4S_5 + S_3C_5 \\
(^2p_6)_x &= C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3 \\
(^2n_6)_y &= S_3(C_4C_5C_6 - S_5S_6) + C_3S_5C_6 \\
(^2s_6)_y &= S_3(-C_4C_5S_6 - S_4C_6) - C_3S_5S_6 \\
(^2a_6)_y &= S_3C_4S_5 - C_3C_5 \\
(^2p_6)_y &= S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3 \\
(^2n_6)_z &= S_4C_5C_6 + C_4S_6 \\
(^2s_6)_z &= -S_4C_5S_6 + C_4C_6 \\
(^2a_6)_z &= S_4S_5 \\
(^2p_6)_z &= S_4S_5d_6
\end{aligned}$$

$$^1\mathbf{A}_6 = \begin{bmatrix} (^1n_6)_x & (^1s_6)_x & (^1a_6)_x & (^1p_6)_x \\ (^1n_6)_y & (^1s_6)_y & (^1a_6)_y & (^1p_6)_y \\ (^1n_6)_z & (^1s_6)_z & (^1a_6)_z & (^1p_6)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.53}$$

where

$$\begin{aligned}
(^1n_6)_x &= C_2[C_3(C_4C_5C_6 - S_4S_6) - S_3S_5C_6] \\
&\quad -S_2[S_3(C_4C_5C_6 - S_5S_6) + C_3S_5C_6] \\
(^1s_6)_x &= C_2[C_3(-C_4C_5S_6 - S_4C_6) + S_3S_5S_6] \\
&\quad -S_2[S_3(-C_4C_5S_6 - S_4C_6) - C_3S_5S_6] \\
(^1a_6)_x &= C_2(C_3C_4S_5 + S_3C_5) - S_2(S_3C_4S_5 - C_3C_5) \\
(^1p_6)_x &= C_2[C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3] \\
&\quad -S_2[S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3] + a_2C_2 \\
(^1n_6)_y &= S_2[C_3(C_4C_5C_6 - S_4S_6) - S_3S_5C_6] + \\
&\quad C_2[S_3(C_4C_5C_6 - S_5S_6) + C_3S_5C_6] \\
(^1s_6)_y &= S_2[C_3(-C_4C_5S_6 - S_4C_6) + S_3S_5S_6]
\end{aligned}$$

$$
\begin{aligned}
& && +C_2[S_3(-C_4C_5S_6 - S_4C_6) - C_3S_5S_6] \\
({}^1a_6)_y &=& & S_2(C_3C_4S_5 + S_3C_5) + C_2(S_3C_4S_5 - C_3C_5) \\
({}^1p_6)_y &=& & S_2[C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3] \\
& && +C_2[S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3] + a_2S_2 \\
({}^1n_6)_z &=& & S_4C_5C_6 + C_4S_6 \\
({}^1s_6)_z &=& & -S_4C_5S_6 + C_4C_6 \\
({}^1a_6)_z &=& & S_4S_5 \\
({}^1p_6)_z &=& & S_4S_5d_6 + d_2
\end{aligned}
$$

Substituting terms from the above matrices, the Jacobian terms relevant for the minimum norm solution may be obtained as

$$
\begin{aligned}
J_{11} &=& & ({}^0p_6)_x({}^0s_0)_x - ({}^0p_6)_y({}^0n_0)_x \\
&=& & -(S_1d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} \\
& && +a_2C_2 + C_1d_6S_4S_5 + d_2) \\
J_{21} &=& & ({}^0p_6)_x({}^0s_0)_y - ({}^0p_6)_y({}^0n_0)_y \\
&=& & C_1d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} \\
& && +a_2C_2 - S_1d_6S_4S_5 + d_2 \\
J_{31} &=& & ({}^0p_6)_x({}^0s_0)_z - ({}^0p_6)_y({}^0n_0)_z \\
&=& & 0.0 \\
J_{12} &=& & ({}^1p_6)_x({}^0s_1)_x - ({}^1p_6)_y({}^0n_1)_x \\
&=& & -(S_2C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3 \\
& && +C_2S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3 + a_2S_2)C_1 \\
J_{22} &=& & ({}^1p_6)_x({}^0s_1)_y - ({}^1p_6)_y({}^0n_1)_y \\
&=& & -(S_2C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3 + \\
& && C_2S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3 + a_2S_2)S_1 \\
J_{32} &=& & ({}^1p_6)_x({}^0s_1)_z - ({}^1p_6)_y({}^0n_1)_z \\
&=& & -(C_2C_3C_4S_5d_6 + S_3(C_5d_6 + d_4) + a_3C_3 \\
& && -S_2S_3C_4S_5d_6 - C_3(C_5d_6 + d_4) + a_3S_3 + a_2C_2) \\
J_{13} &=& & ({}^2p_6)_x({}^0s_2)_x - ({}^2p_6)_y({}^0n_2)_x \\
&=& & -(C_3C_4S_5d_6 + S_3C_5d_6 + d_4 + a_3C_3)C_1S_2 \\
& && -(S_3C_4S_5d_6 - C_3C_5d_6 + d_4 + a_3S_3)C_1C_2 \\
J_{23} &=& & ({}^2p_6)_x({}^0s_2)_y - ({}^2p_6)_y({}^0n_2)_y
\end{aligned}
$$

$$\begin{aligned}
&= -(C_3C_4S_5d_6 + S_3C_5d_6 + d_4 + a_3C_3)S_1S_2 \\
&\quad -(S_3C_4S_5d_6 - C_3C_5d_6 + d_4 + a_3S_3)S_1C_2 \\
J_{33} &= ({}^2p_6)_x({}^0s_2)_z - ({}^2p_6)_y({}^0n_2)_z \\
&= -(C_3C_4S_5d_6 + S_3C_5d_6 + d_4 + a_3C_3)C_2 \\
&\quad +(S_3C_4S_5d_6 - C_3C_5d_6 + d_4 + a_3S_3)S_2 \\
J_{14} &= ({}^3p_6)_x({}^0s_3)_x - ({}^3p_6)_y({}^0n_3)_x \\
&= -C_4S_5d_6S_1 - S_4S_5d_6C_1C_{23} \\
J_{24} &= ({}^3p_6)_x({}^0s_3)_y - ({}^3p_6)_y({}^0n_3)_y \\
&= C_4S_5d_6C_1 - S_4S_5d_6S_1C_{23} \\
J_{34} &= ({}^3p_6)_x({}^0s_3)_z - ({}^3p_6)_y({}^0n_3)_z \\
&= S_4S_5d_6S_{23} \\
J_{15} &= ({}^4p_6)_x({}^0s_4)_x - ({}^4p_6)_y({}^0n_4)_x \\
&= -S_5d_6C_1S_{23} + C_5d_6(C_1C_{23}C_4 - S_1S_4) \\
J_{25} &= ({}^4p_6)_x({}^0s_4)_y - ({}^4p_6)_y({}^0n_4)_y \\
&= -S_5d_6S_1S_{23} + C_5d_6(S_1C_{23}C_4 + C_1S_4) \\
J_{35} &= ({}^4p_6)_x({}^0s_4)_z - ({}^4p_6)_y({}^0n_4)_z \\
&= -S_5d_6C_{23} - C_5d_6S_{23}C_4 \\
J_{16} &= ({}^5p_6)_x({}^0s_5)_x - ({}^5p_6)_y({}^0n_5)_x \\
&= 0.0 \\
J_{26} &= ({}^5p_6)_x({}^0s_5)_y - ({}^5p_6)_y({}^0n_5)_y \\
&= 0.0 \\
J_{36} &= ({}^5p_6)_x({}^0s_5)_z - ({}^5p_6)_y({}^0n_5)_z \\
&= 0.0
\end{aligned} \tag{A.54}$$

## A.5 Joint Torques for PUMA-560

The three important techniques for calculating the joint torques and forces are the Lagrange-Euler formulation, the Newton-Euler formulation and the generalized D'Alembert equations of motion. The method described below (and implemented) is based on the Newton-Euler formulation and is due to Luh et. al. [88]. This method has been chosen because it's easy to cast it into a computer algorithm, and is fast enough for real-time implementation.

Let ${}^{i-1}\mathbf{R}_i$ be a $3\times3$ rotation matrix which transforms any vector with reference to coordinate frame $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$ to the coordinate system $(\mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{z}_{i-1})$. This is the upper left $3 \times 3$ submatrix of ${}^{i-1}\mathbf{A}_i$. Thus

$$ {}^{i-1}\mathbf{R}_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix} \tag{A.55} $$

and

$$ {}^{i}\mathbf{R}_{i-1} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ -\cos\alpha_i \sin\theta_i & \cos\alpha_i \cos\theta_i & \sin\alpha_i \\ \sin\alpha_i \sin\theta_i & -\sin\alpha_i \cos\theta_i & \cos\alpha_i \end{bmatrix} \tag{A.56} $$

Further let us define the following quantities:

| | | |
|---|---|---|
| ${}^i\mathbf{R}_0\omega_i$ | = | angular velocity of link $i$ with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\dot{\omega}_i$ | = | angular acceleration of link $i$ with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| $m_i$ | = | total mass of link $i$ |
| ${}^i\mathbf{R}_0\bar{\mathbf{s}}_i$ | = | centre of mass of link $i$ referred to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\mathbf{p}_i^*$ | = | location of $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$ from the origin of $(\mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{z}_{i-1})$ with respect to the $i^{th}$ coordinate frame. |
| | = | $\begin{bmatrix} a_i \\ d_i \sin\alpha_i \\ d_i \cos\alpha_i \end{bmatrix}$ |
| $({}^i\mathbf{R}_0\mathbf{I}_i^0\mathbf{R}_i)$ | = | the inertia matrix of link $i$ about its centre of mass referred to its own link coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\dot{\mathbf{v}}_i$ | = | linear acceleration of link $i$ with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\bar{\mathbf{a}}_i$ | = | linear acceleration of the centre of mass of link $i$ with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\mathbf{F}_i$ | = | total external force on link $i$ at its centre of mass with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\mathbf{N}_i$ | = | total external moment on link $i$ at its centre of mass with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |

| | | |
|---|---|---|
| ${}^i\mathbf{R}_0\mathbf{f}_i$ | = | force exerted on link $i$ by link $i-1$ at the coordinate frame $(\mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{z}_{i-1})$ to support link $i$ and the links above it referenced with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| ${}^i\mathbf{R}_0\mathbf{n}_i$ | = | moment exerted on link $i$ by link $i-1$ at the coordinate frame $(\mathbf{x}_{i-1}, \mathbf{y}_{i-1}, \mathbf{z}_{i-1})$ referenced with respect to its own coordinate system $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$. |
| $\tau_i$ | = | input force/torque at joint $i$. |
| $q_i$ | = | value of joint variable $i$. |
| $\dot{q}_i$ | = | rate of change of joint variable $q_i$. |
| $\ddot{q}_i$ | = | rate of change of joint variable $\dot{q}_i$. |

It is known that the Newton-Euler dynamic equations can be formulated as a set of compact forward and backward recursive equations. This set of recursive equations can be applied sequentially to the robot links. The forward recursion propagates kinematics information such as angular velocities, angular accelerations, and linear accelerations from the base reference frame (inertial frame) to the end-effector. The backward recursion propagates the forces exerted on each link from the end-effector of the manipulator to the base frame, and the applied joint torques are computed from these forces.

The forward equations for revolute joints are reproduced below.

**Forward Equations** : $i = 1, 2, \ldots, n$

$$
\begin{aligned}
{}^i\mathbf{R}_0\omega_i &= {}^i\mathbf{R}_{i-1}({}^{i-1}\mathbf{R}_0\omega_{i-1} + \mathbf{z}_0\dot{q}_i) && \text{(A.57)}\\
{}^i\mathbf{R}_0\dot{\omega}_i &= {}^i\mathbf{R}_{i-1}[{}^{i-1}\mathbf{R}_0\dot{\omega}_{i-1} + \mathbf{z}_0\ddot{q}_i + ({}^{i-1}\mathbf{R}_0\omega_{i-1}) \times \mathbf{z}_0\dot{q}_i] && \text{(A.58)}\\
{}^i\mathbf{R}_0\dot{\mathbf{v}}_i &= ({}^i\mathbf{R}_0\dot{\omega}_i) \times ({}^i\mathbf{R}_0\mathbf{p}_i^*) + ({}^i\mathbf{R}_0\omega_i) \times [({}^i\mathbf{R}_0\omega_i) \times ({}^i\mathbf{R}_0\mathbf{p}_i^*)] \\
&\quad + {}^i\mathbf{R}_{i-1}({}^{i-1}\mathbf{R}_0\dot{\mathbf{v}}_{i-1}) && \text{(A.59)}\\
{}^i\mathbf{R}_0\bar{\mathbf{a}}_i &= ({}^i\mathbf{R}_0\dot{\omega}_i) \times ({}^i\mathbf{R}_0\bar{\mathbf{s}}_i) + ({}^i\mathbf{R}_0\omega_i) \times [({}^i\mathbf{R}_0\omega_i) \times ({}^i\mathbf{R}_0\bar{\mathbf{s}}_i)] + {}^i\mathbf{R}_0\dot{\mathbf{v}}_i \\
& && \text{(A.60)}
\end{aligned}
$$

The backward equations are given below. **Backward equations** : $i = n, n-1, \ldots, 1$

$$
\begin{aligned}
{}^i\mathbf{R}_0\mathbf{f}_i &= {}^i\mathbf{R}_{i+1}({}^{i+1}\mathbf{R}_0\mathbf{f}_{i+1}) + m_i^i\mathbf{R}_0\bar{\mathbf{a}}_i && \text{(A.61)}\\
{}^i\mathbf{R}_0\mathbf{n}_i &= {}^i\mathbf{R}_{i+1}[{}^{i+1}\mathbf{R}_0\mathbf{n}_{i+1} + ({}^{i+1}\mathbf{R}_0\mathbf{p}_i^*) \times ({}^{i+1}\mathbf{R}_0\mathbf{f}_{i+1})] + ({}^i\mathbf{R}_0\mathbf{p}_i^* + {}^i\mathbf{R}_0\bar{\mathbf{s}}_i) \times \\
&\quad ({}^i\mathbf{R}_0\mathbf{F}_i) + ({}^i\mathbf{R}_0\mathbf{I}_i^0\mathbf{R}_i)({}^i\mathbf{R}_0\ddot{\omega}_i) + ({}^i\mathbf{R}_0\omega_i) \times [({}^i\mathbf{R}_0\mathbf{I}_i^0\mathbf{R}_i)({}^i\mathbf{R}_0\omega_i)] && \text{(A.62)}\\
\tau_i &= ({}^i\mathbf{R}_0\mathbf{n}_i)^T({}^i\mathbf{R}_{i-1}\mathbf{z}_0) + b_i\dot{q}_i && \text{(A.63)}
\end{aligned}
$$

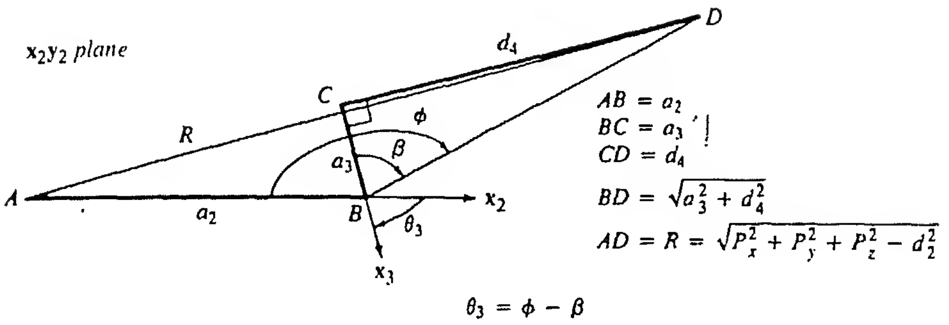
where $\mathbf{z}_0 = (0,0,1)^T$ and $b_i$ is the viscous damping coefficient for joint $i$. The usual initial conditions are $\omega_0 = \dot{\omega}_0 = \mathbf{v}_0 = 0$ and $\dot{\mathbf{v}}_0 = (g_x, g_y, g_z)^T$ (to include gravity), where $|\,\mathbf{g}\,| = 9.8062\mathrm{m/s}^2$.

The torque calculations are done according to the algorithm given below.

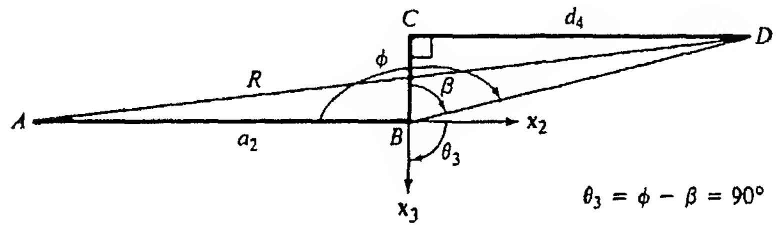
```
procedure FindTorques;
   begin
     n := number of links;
     $\omega_0$ := 0;
     $\dot{\omega}_0$ := 0;
     $\mathbf{v}_0$ := 0;
     $\dot{\mathbf{v}}_0$ := g;
     i := 1;
     for i := 1 to n do
       begin
         compute ${}^i\mathbf{R}_0\omega_i$;
         compute ${}^i\mathbf{R}_0\dot{\omega}_i$;
         compute ${}^i\mathbf{R}_0\dot{\mathbf{v}}_i$;
         compute ${}^i\mathbf{R}_0\bar{\mathbf{a}}_i$;
       end;
     $\mathbf{f}_{n+1}$ := external force;
     $\mathbf{n}_{n+1}$ := external moment;
     for i := n downto 1 do
       begin
         compute ${}^i\mathbf{R}_0\mathbf{F}_i$;
         compute ${}^i\mathbf{R}_0\mathbf{N}_i$;
         compute ${}^i\mathbf{R}_0\mathbf{f}_i$;
         compute ${}^i\mathbf{R}_0\mathbf{n}_i$;
         compute $\tau_i$;
       end;
   end;
```

The inertia parameters of PUMA-560 referenced with respect to the coordinate system established in Figure A.1 are given in Table A.2. (Source [125][1])

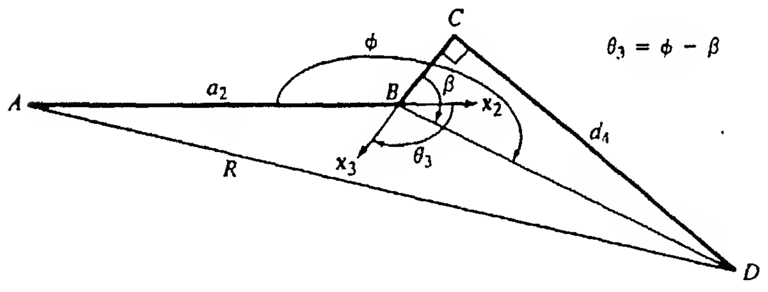
[1]Some values have been modified from this source to tally with the coordinate system we have chosen



Left and below arm



Left and below arm



Left and above arm

Figure A.5: Solution for joint 3 of PUMA-560

| Inertia parameters for PUMA-560 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Centre of mass | | | Radius of gyration | | |
| Joint $i$ | Mass $m_i$ (Kg) | $\bar{x}_i$ (m) | $\bar{y}_i$ (m) | $\bar{z}_i$ (m) | $(k_i)_{xx}$ (m) | $(k_i)_{yy}$ (m) | $(k_i)_{zz}$ (m) |
| 1 | 12.96 | 0.00 | 0.3088 | 0.0389 | 0.1816 | 0.0152 | 0.1811 |
| 2 | 22.37 | −0.3289 | 0.0050 | 0.2038 | 0.5696 | 0.1930 | 0.1514 |
| 3 | 5.01 | 0.0204 | 0.0137 | 0.0037 | 0.0151 | 0.0155 | 0.0021 |
| 4 | 1.18 | 0.0 | 0.0863 | −0.0029 | 0.0119 | 0.0029 | 0.0118 |
| 5 | 0.62 | 0.0 | −0.0102 | 0.0013 | 0.0009 | 0.0009 | 0.0009 |
| 6 | 0.16 | 0.0 | 0.0 | 0.0029 | 0.0008 | 0.0008 | 0.0004 |

Table A.2: Inertia parameters for the PUMA-560 robot

# Appendix B

# Objects Describing Mathematical Classes

The following appendices list some of the header files needed for defining objects. The purpose of presenting the header files is to highlight the object-oriented implementation aspects of modelling a robot. The code is given in C++ and has been implemented in the DOS environment.

## B.1 Points

Two classes are defined to describe a point — one in 2-D space and another in 3-D space. Since this class is needed by all classes, it is given public access by declaring it as `struct`.

```
// ********   OBJECTS DESCRIBING POINTS IN SPACE  ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

#define sign(x)  ((x >= 0) ? (1) : (-1))
#define abs_(a)  (((a) >= (0.0)) ? (a) : (-a))
#define round(x)  ((sign(x))*(int)((abs_(x))+(0.5)))
enum Boolean {false, true};
```

```
struct point_2D {
   int XX,YY;
};
struct point_3D {
   int XXX, YYY, ZZZ;
public:
   point_3D() { XXX=0, YYY=0; ZZZ=0; };  // default constructor
   point_3D(int ex, int yee, int zed){ XXX=ex; YYY=yee; ZZZ=zed;};
   double separation(point_3D P); // separation between two points
   point_3D operator+(point_3D& pp); // adding two points
};
```

## B.2 Vectors

A vector is a common object needed for representing rays, positions, escape vectors, and a host of other data needed for inverse kinematics and dynamics. The common operations on vectors are vector addition, vector subtraction, dot product, cross product, angle between two vectors, scalar multiplication, translation of a point through a vector and rotation of a vector towards another.

```
// ********   OBJECT DESCRIBING A VECTOR    ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

const size = 3;
class Vector {
   float vv[size];
   void error(char *msg);
public:
   Vector(float initval);
   Vector();
   Vector(float f1, float f2, float f3);
   Vector(point_3D point);
   Vector(point_3D pfrom, point_3D pto);
   float Mag();
   void Reset(float initval=0.0);
   float& operator[](int index);
   const float *operator&() { return vv; };
   float operator*() { return vv[0]; };
   float operator() (int el, float f);
   Vector operator*(float);    //  scalar multiplication
   Vector operator+();     //  return a unit vector
   Vector operator&(Vector& vec);   //  cross product
   float operator%(Vector& vec);   //  dot product
   float operator^(Vector& vec);   //  angle between two Vectors
   Vector operator+(Vector& vec);   // vector addition
   point_3D operator+(point_3D& pp); // translate a point
   Vector operator-(Vector& vec);   // vector subtraction
   Vector operator+=(Vector& vec);   // vector addition
   Vector operator-=(Vector& vec);   // vector subtraction
   Vector operator*=(float);    // scalar multiplication
   Vector operator-();     // unary minus
   void print(char *msg = "");
};
point_3D PointTowards(point_3D& from, point_3D& towards, float length);
point_3D PointTowards(point_3D& from, Vector& vec, float length);
Vector Rotate(Vector& from, Vector& towards, float byangle);
```

## B.3 Planes

A plane is the bounding surface for obstacles, links and grasp objects. The boundary of a plane is assumed to be a quadrilateral (a polygon in general). The normal vector is defined according to the naming convention described in Section 6.6.1. The function `Pierces` finds the point on the plane where a vector originating from a point intersects the plane (See Section 6.6.1).

```
// ********   OBJECT DESCRIBING A 3-D PLANE  ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class Plane {
   Vector normal;     // the normal vector to the plane
   point_3D corner[4];  // the four points that define the plane
public:
   Plane(point_3D P1, point_3D P2, point_3D P3, point_3D P4);
   Plane();
   void FindNormalVector();
   point_3D Pierces(Vector& front, point_3D& from);
   friend Obstacle;
   friend Sensor;
};
```

## B.4 Transformations

The D-H transformation is a matrix class with a fixed dimension, viz. 4. The transformation class follows all normal matrix operations like addition, subtraction and multiplication. Operations involving scalar quantities are also supported in class Transformation. Two special assignments are defined according to the transformation matrix defined in (A.1) and (A.2). Another special assignment is based on the position of the grasped object.

```
// ******   OBJECT DESCRIBING A D-H TRANSFORMATION    ****** //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

const SIZE=4;

class Transformation { // the D-H transformation matrix
   double m[SIZE][SIZE]; // make a 4x4 matrix
   double *Ai, *Di, *Alphi, *Thetai;
   void error(char *msg1, char *msg2 = "");
public:
   Transformation(double *ai, double *di,
                  double *alphi, double *thetai);
   Transformation();
   Transformation(const Transformation& trans);
   ~Transformation();
   void SetParameters(double *ai, double *di, double *alphi,
                      double *thetai);
   void SetDiagonal(double zero0, double one1, double two2);
   Transformation operator+(const Transformation &rval);
        // matrix addition
   Transformation operator+(const double rval);
        // scalar addition
   Transformation operator-(const Transformation &rval);
        // matrix subtraction
   Transformation operator-(const double rval);
```

```
        // scalar subtraction
   Transformation operator-();  // unary minus
   Transformation operator*(const Transformation &rval);
        // matrix multiplication
   Transformation operator*(const double rval);
        // scalar multiplication
   Transformation Transformation::operator*=
     (const Transformation &arg); // matrix multiplication
   Vector operator*(const Vector& vec);
        // vector multiplication
   point_3D Transformation::operator*(const point_3D &pnt);
        // transformation of a point
   double &val(int row, int col); // element selection
   void Update();
   void UpdateToInverse();
   void GripObjectUpdate(int px, int py, int pz);
       // used for grasped object
   double &mval(int row, int col)
   { return (m[row][col]); }
   // can be used both to read and write an element
   // used by matrix functions which KNOW they aren't
   // exceeding the boundaries
   void print(char *msg="");
};
```

## B.5 Fuzzy Numbers

A fuzzy number is a special kind of set having a grade of membership for each element. There are two universe types, viz Distance and Angle. The generalized Π-function given in (2.24) has two special forms, the $S^-$ function (also called the $Z$-function) and the $S^+$-function (also called simply $S$-function). The commonly used linguistic modifiers (see Section 2.4.2) are the concentration and dilation operation. These are also called hedges [137]. The `class Fuzzy` includes functions for generating a fuzzy number from the membership function, for defuzzifying a fuzzy number, changing the shape of a fuzzy number by applying hedges, rotating a fuzzy number (needed for frame rotation as in Section 4.3.2), generating the elements of the universe of distance with increasing gradation and finding the composition of two fuzzy numbers.

```
// ********   OBJECT DESCRIBING A FUZZY NUMBER    ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

#define max(a,b)     (((a) > (b)) ? (a) : (b))
#define min(a,b)     (((a) < (b)) ? (a) : (b))

enum UniverseType {
   Distance, Angle
};

enum FunctionType {
   PI_function, S_function, Z_function
};

enum Hedge {
   CON, DIL
};

class Fuzzy {
   FunctionType meaning;
   char term[20];
   void print();
public:
   UniverseType universe;
   int possibility[16];
   int univsize;
   Fuzzy(char *name="",UniverseType univ=Distance,
      FunctionType func=PI_function, int leftval=0,
      int centreval=175, int rightval=HIGH_DISTANCE);
   ~Fuzzy();
   float Defuzzified(Fuzzy fnum);
   void Generate(FunctionType functionname, double leftlimit,
     double centralvalue, double rightlimit );
   Fuzzy operator+=(const Fuzzy& arg);
         // finds the resultant of two numbers
   void Rotate(int RotateUnits);
   void Modify(Hedge modifiername);
   double CrispAngle(double *Xmotion, double *Ymotion);
   float CrispDistance();
   int FrameRotation();
};

// Some other general function prototypes used by
// the class functions of Fuzzy
void Composition(int RuleNumber, const Fuzzy& num1,
     const Fuzzy& num2, Fuzzy& result);
void ForgetRules();
void FormRule();
void GenDistUniverse(void);
void DefineFuzzyQuantifiers(void);
double pi_x(const double& u, const double& p, const double& beta);
```

Appendix C

# Objects Describing Environment Parameters

## C.1 Path

The definition of spatial paths is needed for describing the end-effector trajectory and the obstacle trajectory for simulation runs. A spatial curve is represented in parametric form. The position of a point is ascertained by examining the value of the variable 'parameter'. This class also includes functions for determining the "best" approach direction of the end-effector in the absence of any other geometric information pertaining to the manipulator.

```
// ********   OBJECT DESCRIBING A SPATIAL CURVE   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class Path {
   float parameter; // decides position on path : varies from 0 to 1
   float increment; // value by which 'parameter' must be incremented
   int segments; // tells how many segments the path consists of
   // (number of points = segments + 1)
   point_3D beginpoint, endpoint;
   // terminal points of the current segment
   point_3D currentpoint;
   point_3D joinpoint[5];
   // is the point at which the segments are connected
   int currentseg; // the current segment number of path
   int pointcount, skippoints;
public:
   Path(point_3D start,point_3D finish,float par=0.0,float inc=0.05) {
        beginpoint = currentpoint = joinpoint[0] = start;
        endpoint = joinpoint[1] = finish;
        parameter = par;
        increment = inc;
        segments = 1;
        skippoints = pointcount = currentseg = 0;
   };
   Path(int bx, int by, int bz, int ex, int ey, int ez,
        float par=0.0, float inc=0.05);
   Path(char pathfilename[15]);
   void NewPath(point_3D start, point_3D finish, float par=0.0) {
        beginpoint = currentpoint = joinpoint[0] = start;
        endpoint = joinpoint[1] = finish;
        parameter = par;
        segments = 1;
        skippoints = pointcount = currentseg = 0;
   };
   void SetParameter(float par=0.0, float inc=0.05) {
        parameter = par;
        increment = inc;
   };
   void SetParameter(float par=0.0) { parameter = par; };
   void ChangeIncrement(float magnifyby=0.5)
        { increment *= magnifyby; };
   float GetParameter() { return parameter; };
   int funcX(float par);
   int funcY(float par);
   int funcZ(float par);
   int funcTangentX(float par);
   int funcTangentY(float par);
   int funcTangentZ(float par);
   point_3D CurrentPoint();
   point_3D PathPoint(float value);
   point_3D NextPoint();
   point_3D PreviousPoint();
   Boolean end();  // determines end of path
   Vector Tangential();
   point_3D BaseApproachPoint(point_3D& towards, float length);
   point_3D VerticalApproachPoint(float length);
   point_3D VectorDirPoint(point_3D& towards, float length);
};
```

## C.2 Obstacles

The obstacle, assumed to be a polyhedron, is defined by its vertices. The set of vertices forming a surface are stored and adjacent surfaces are marked. This object class stores the coordinates of all its vertices and is responsible for updating its vertex positions when the obstacle moves.

```
// ********   OBJECT DESCRIBING AN OBSTACLE   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class Obstacle {
   int X, Y, Z;   // position of reference centre of obstacle
   int numvertices, numedges, numsides;
   struct {
      int v1, v2;
   } edge[35];
   struct {
      int p1, p2, p3, p4;
   } surface[15];
   point_3D cornerL[25];
    //  corner vertices w.r.t link (user assigns it)
   FILE *obstdatafile;
   Boolean isOpen;
   int readwritecalls;
public:
   point_3D cornerW[25]; //  corner vertices in world coordinates
   point_2D projcorner[25]; // projected corner vertices inside window
   Plane side[15];      // the sides of the obstacle are planes
   Obstacle(int X=0, int Y=0, int Z=0);
   void Put(int newX, int newY, int newZ);
   void Put(point_3D newpoint);
   void Move(Path& tracepath);
   int FindVertices();
   void OpenDataFile(char *fname);
   void CloseDataFile();
   void ReadCorners();
   void ReadSides();
   void ReadObstacleData(char *diskfilename);
   void WriteObstaclePath(char *diskfilename);
   void ReadPath(char *diskfilename);
   int FindNumEdges();
   int FindNumSides();
   void ReadAllEdges();
   void RelocateCorners();
   void AssignPlanes();
   friend class Sensor;
   friend class Window;
};
```

## C.3 Sensors

A sensor is fixed on a link or a body and knows its position relative to the base coordinate of the parent link. Its absolute position in space and the absolute direction of the normal vector is obtained from the absolute orientation of the link. This class includes functions for determining whether a sensor intercepts an obstacle or a link.

```
// ********   OBJECT DESCRIBING A SENSOR    ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class Sensor {
   point_3D linkPos;
    // position of sensor w.r.t. Link
   point_3D absPos;
    // absolute position of above point in world reference
   point_3D linkNormal;
```

```
    // point outside link which defines Normal w.r.t Link
   point_3D absNormal;
    // absolute position of above point in world coordinates
   Vector vecNormal;
    // vector defining Normal in world coordinates
   float beamangle;     // in radians
public:
   Boolean IsActive;    // has the sensor detected an obstacle ?
   Sensor(int lX=0, int lY=0, int lZ=0, float angle=90.0);
       // angle in degrees
   Sensor();
   void FixLinkLocation(int sX, int sY, int sZ);
   void FixAbsLocation(int aX, int aY, int aZ);
   void FormNormalVector();
   Boolean Intercepts(Obstacle& object, Vector& escapedir);
   Boolean Intercepts(Linktype& object, Vector& escapedir);
   friend class LinkType;
   friend class Robot;
   friend class Window;
   friend class Obstacle;
};
```

## Appendix D

# Objects Describing the Robot

A robot consists of links and the grasp body. The grasp body is special kind of link because it coalesces with the gripper when the robot is holding the object. Two kinds of objects are thus described before the description of a robot is taken up — a link and a grasp body.

## D.1 Links

A link has certain parameters associated with it, namely $a$, $d$ and $\alpha$ and other kinematic parameters which describe the upper and lower bounds on the joints and the maximum velocity and acceleration. The shape of the link is stored through a set of vertices which form surfaces and then a volume. The dynamic parameters like radius of gyration, force and moment etc. are stored as vectors. The matrix defining the D-H transformation is stored as `class Transformation`. This class has a few functions to read data from an ASCII file.

```
// ********   OBJECT DESCRIBING A LINKTYPE   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class LinkType {
   double d, a, alpha;  // alpha is in DEGREES
   float lolim, uplim;  // both in DEGREES
   float angVelmax, angAccmax;
    // in 'rad/sec' and 'rad/sec/sec' respectively
   int numvertices, numedges, numsensors;
   struct {
      int v1, v2;
   } edge[35];
   // these are indices of points for each line in the wire-frame
   point_3D cornerL[25]; //  corner vertices w.r.t link frame
   FILE *linkdatafile;
   Boolean isOpen;
   struct {
      int p1, p2, p3, p4;
   } surface[15];
   // all following parameters used in inverse dynamics
   float mass, damping, maxTorque;
   Vector rGyration;
   Vector Romega, Romegadot, Rvdot;
   Vector Rpstar, Rabar, Rsbar;
   Vector Rforce, Rmoment;
public:
   Sensor ulsonic[7];
   Transformation linktrans;
   Plane side[15]; // are the bounding planes of the link
   point_3D cornerW[25];
    //  corner vertices in world coordinates
   point_2D projcorner[25];
    //  projected corner vertices in window frame
   double theta;
    //  stored in radians
   double thetadot, thetadotdot;
    //  in 'rad/sec' and 'rad/sec/sec' respectively
   LinkType(double dd=0, double aa=0, double al=0, float lo=0,
    float up=180, float avel=1, float aacc=1);
   DynamicParameters(float ms=0, float dmp=0, float maxT=50,
                     Vector& rG, Vector& Rsb);
   point_3D Origin();
   int FindVertices();
   void OpenDataFile(char *fname);
   void CloseDataFile();
   void ReadCorners();
   int FindNumEdges();
   void ReadAllEdges();
   void ReadSensors();
   void ReadLinkData(char *diskfilename);
   friend class Robot;
   friend class Window;
};
```

## D.2 Grasp Body

The grasp body is a special kind of link which behaves in two ways depending on whether it is being held by the gripper. When the gripper is free, the grasp body behaves as a stationary object. When the gripper is holding the grasp body, it becomes an integral part of the robot and behaves as an extension of the end-effector. This class also includes a few functions for reading data from an ASCII file.

```
// ********   OBJECT DESCRIBING A GRASP BODY   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class Graspbody {
   double d, a, alpha;  // alpha is in DEGREES
   int numedges, numvertices;
   struct {
      int v1, v2;
   } edge[20];
   // these are indices of points for each line in the wire-frame
   point_3D cornerL[25];  // corner vertices w.r.t local origin
   // the point cornerL[numvertices+1] is the position
   // of the local origin
   struct {
      int p1, p2, p3, p4;
   } surface[15];
   FILE *graspdatafile;
   Boolean isOpen;
   point_3D initApproach, finalApproach;
   point_3D initOrigin, finalOrigin;
   Vector finalX, finalY, finalZ;
   float mass;
public:
   Transformation linktrans;
   point_3D cornerW[25]; //  corner vertices in world coordinates
   point_2D projcorner[25]; //  projected corner vertices in window
   Plane side[15];      //  the bounding planes
   double theta;     //  stored in radians
   Graspbody(double dd=0.0, double aa=0.0, double al=0.0);
   int FindVertices();
   void OpenDataFile(char *fname);
   void CloseDataFile();
   void ReadCorners();
   int FindNumEdges();
   void ReadAllEdges();
   point_3D Origin();
   Vector Xaxis();
   Vector Yaxis();
   Vector Zaxis();
   void ReadGraspData(char *diskfilename);
   friend class Robot;
   friend class Window;
   friend void WriteJointValues(Robot& rob);
   friend int ReadJointValues(Robot& rob);
};
```

## D.3 Robot

Physically speaking, a robot consists of a chain of links and a grasp body. The other parameters that are of significance are the maximum links of the robot, the state of the manipulator, the state of the gripper, the kind of task to be performed – whether point-to-point or path-preference, the location of the robot base, the location of the 'pivot points', the escape vectors at the pivot points, the hand vectors $[\vec{n}, \vec{s}, \vec{a}]$ and others. The functions include solution of the direct kinematic problem, solution of the inverse kinematic problem, the minimum norm solution for positional control in task space, the inverse dynamic solution and a host of other routines for moving the manipulator towards the goal position according to the *state* of the manipulator.

```
// ********   OBJECT DESCRIBING A ROBOT   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

enum ArmIndicator {
   on = 1,
   off = -1
};
enum ManipState {
   NATURAL=1, PAUSE, DEADLOCK, SURRENDER, DETOUR,
   ADVANCE, ORIENTING, APPROACHING, RETRACTING
};
enum GraspState {
   EMPTY=1, HOLDING
};
enum TaskType {
   PointToPoint, PathPreference
};

class Robot {
   char Make[20];    // is the common name of the robot
   int maxlinks;     // maximum number of links
   char robotfile[20]; // name of the file to write joint values
   LinkType *link[8];
   Graspbody *graspobject;
   double A2, D2, A3, D4, C1, C2, C3, S1, S2, S3,
          C23, S23, S4, C4, S5, C5, S6, C6;
   // these are some variables used for short-form notation
   point_3D robotbase; // the base of the robot
   point_3D parkposition, endparkposition;
     // are the parking positions
   float OO, AA, TT;
     // the orientation vector of end-effector (specific to PUMA)
   float toollength;  // the length of tool (including d6)
   Vector shoulderEscape; // the escape vector acting at 'shoulder'
   Vector elbowEscape;    // the escape vector acting at 'elbow'
   Vector wristEscape;    // the escape vector acting at 'wrist'
   Vector fingerEscape;   // the escape vector acting at 'EE'
   Vector N;    // normal vector of the hand   (unit vector)
   Vector S;     // sliding vector of the hand  (unit vector)
   Vector A;     // approach vector of the hand (unit vector)
   void AssignAbbreviations();
   Boolean jobStatus;
   TaskType PTP;
public:
   ArmIndicator ARM, ELBOW, WRIST, FLIP;
      // the robot configuration parameters
   ManipState MState;
   GraspState GState;
   point_3D shoulder; // stores the shoulder location
   point_3D elbow;    // stores the elbow location
   point_3D wrist;    // stores the wrist position
   point_3D EE;    // stores the end-effector location
   Robot(char *robotmake="PUMA 560", int maxl=8,
     float tool=120.0, char *jointfilename="ROBOT.JNT");
     // base is included in total links, actually (1,2,...,6)
   ~Robot();
   void ParkPosition(int parkx, int parky, int parkz);
   void ParkPositions(point_3D beginparkxyz, point_3D endparkxyz);
   void AssignLink(int lnum, LinkType *lt);
   void PlaceGripObject(Grippable *go, point_3D& startXYZ,
        point_3D& endXYZ);
   double& Joint(int jnt) { return link[jnt]->theta; };
   void SetJoint(int jnt, double value);
   void SetLinkParameters();
   void SetTaskType(TaskType tt) { PTP = tt; };
   Boolean IsPointToPoint()
   { return (PTP == PointToPoint) ? true : false; };
   void Reposition(int lnum, point_3D& wrtworld, point_3D wrtlink);
   Transformation CurrentTransMatrix(int linknum);
   void RelocateLinks(int withgripobject);
   void RelocateGraspObject(Transformation& transmat);
   void LocateGraspObject();
   void ArmSolution();
   void WristSolution();
   void Convert_OAT();
   void TrigArmSolution();
   void DetectObstacle(Obstacle& hurdle);
```

```
    point_3D TipPosition(double D6);
    void FindTransformation(Path& curve);
    void FindTransformation(Vector& xaxis, Vector& yaxis,
        Vector& zaxis);
    void DirectKinematics();
    void MoveOn(Path& EEpath, int option);
    void MoveTo(point_3D& endposition, int option);
    void MoveTowards(point_3D& towards, int option);
    void GeometricSolution();
    int MinimumNorm(point_3D nextEEpos);
    void DecideManipState();
    void DecideGraspState();
    int CheckJointVelAcc();
    point_3D NextGoalPoint();
    Boolean TaskAccomplished();
    friend class LinkType;
    friend class Window;
    friend void WriteJointValues(Robot& rob);
    friend int ReadJointValues(Robot& rob);
};
```

Appendix E

# Objects for Graphic Display

There are two kinds of graphic objects. Class `ViewParameters` is used for setting the viewing parameters and another class `Window` inherits the previous class.

## E.1 Viewing Parameters

The viewing parameters like reference centre, the normal to the viewing plane, the viewing direction, the centre of projection, the centre of the viewport and others are stored in `class ViewParameters`. It also includes functions for setting these parameters and doing matrix transformations for projection of a three dimensional point onto a two-dimensional screen. The details of the functions may be obtained from [53].

```
// ********   OBJECT DESCRIBING VIEWING PARAMETERS   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class ViewParameters {
    double roundoff;
protected:
    float refX, refY, refZ;
    float normDX, normDY, normDZ;
    float viewD;
    float upDX, upDY, upDZ;
    float projDX, projDY, projDZ;
    Boolean perspective;
    float centerXP, centerYP, centerZP;
    float VPcenterXP, VPcenterYP, VPcenterZP;
    float sXP, sYP;
    float T[4][3];
    char viewparfilename[20];
public:
    ViewParameters(int Xref=0, int Yref=0, int Zref=0,
        int viewD=0, int upDX=0, int upDY=1, int upDZ=0,
        int parperDX=1, int parperDY=1, int parperDZ=1,
        int normDX=-1, int normDY=-1, int normDZ=-1);
    void NewViewParameters(int Xref=0, int Yref=0, int Zref=0,
        int viewD=0, int upDX=0, int upDY=1, int upDZ=0,
        int parperDX=1, int parperDY=1, int parperDZ=1,
        int normDX=-1, int normDY=-1, int normDZ=-1);
    void ReadViewParameters(char *filename);
    void SaveViewParameters();
    void NewTransform();
    void Translate3(float TX, float TY, float TZ);
    void RotateX3(float S, float C);
    void RotateY3(float S, float C);
    void RotateZ3(float S, float C);
    void SetViewReference(int X, int Y, int Z);
    void SetViewPlaneNormal(int DX, int DY, int DZ);
    void SetViewDistance(int D);
    void SetViewUp(int DX, int DY, int DZ);
    void SetParallel(int DX, int DY, int DZ);
    void SetPerspective(int X, int Y, int Z);
    void MakeViewPlaneTrans();
    void MakePerspectiveTrans();
    void MakeParallelTrans();
    void PerspectiveTrans(float& X, float& Y, float& Z);
    void ParallelTrans(float& X, float& Y, float& Z);
    void ViewPlaneTransform(float& XC, float& YC, float& ZC);
    void ProjectPoint(float X, float Y, float Z,
                      int& imgX, int& imgY);
};
```

## E.2 Window

The class Window inherits the viewing parameters described above and has other routines for opening and closing a window on any part of the screen and drawing on that window. It also has routines for opening a plotter file and sending output to that file.

```
// ********   OBJECT DESCRIBING A WINDOW   ********* //
// Author : Anupam Bagchi
// Robotics Centre, IIT Kanpur, 208 016, INDIA, (November 1991)

class far Window : public Quadpair, public ViewParameters {
   Boolean isOpen;
   int VportX1, VportY1, VportX2, VportY2;
   int VportXspan, VportYspan;
   int drawcolor;
public:
   Window(int Xleft=0, int Ybottom=0, int Xright=640, int Ytop=350);
   ~Window();
   void far Open(char *header, int top_font, int top_font_color,
                 int top_font_size, int border_color);
   void far Close();
   void far Clear();
   void setVport(int VportXleft=0, int VportYtop=0, int Xwidth=640,
                 int Yheight=350);
   int GetXlow();
   int GetXhigh();
   int GetYlow();
   int GetYhigh();
   void far MapToVport(int *VpX, int *VpY, int WinX, int WinY);
   void far draw(int X1, int Y1);
   void far draw(int X1, int Y1, int X2, int Y2);
   void far draw(int xcen, int ycen, int radius);
   void far draw(Sphere& object);
   void far draw(Obstacle& object);
   void far draw(Path& curve, int color=WHITE);
   void far draw(Robot& manipulator);
   void far ProjectLink(LinkType& linkbody);
   void far ProjectGripObject(Grippable& gripbody);
   void far plotGripObject(Grippable& gripbody);
   void far OpenPlotter(int Xbase, int Ybase, int Xwidth,
                        int Ywidth, char *header);
   void far ClosePlotter();
   void far plot(int X1, int Y1, int X2, int Y2); // plot a line
   void far plot(point_3D& p1, point_3D& p2); // plot a line in space
   void far plot(Obstacle& object);   // plot an obstacle-like object
   void far plot(Path& curve);   // plot a path
   void far plotLink(LinkType& linkbody);
   void far plot(Robot& manipulator);
   void far MarkSensors(Robot& manipulator);
   void SetColor(int color);
};

void RoundedRectangle(int xmin, int xmax, int ymin, int ymax,
                      int radius);

// ********   QUADPAIR IS INHERITED BY WINDOW   ********* //

class Quadpair {
protected: // allows derived class to access private data
   int X1;
   int Y1;
   int X2;
   int Y2;
public:
   Quadpair(int InitX1, int InitY1, int InitX2, int InitY2);
   int GetX1();
   int GetY1();
   int GetX2();
   int GetY2();
};
```

Appendix F

# 3-D Model of PUMA-560 and Placement of Sensors

## F.1 Link Shapes Used for Modelling

All links surfaces are assumed to comprise of planes. Planes are defined by a set of edges forming a closed loop. The edges in turn, are defined by vertices. In this Appendix the link shapes used for modelling the PUMA-560 are presented. The dimensions are close to the actual values, but since curvatures have been ignored, the resulting shape may best be called close to approximate. The placement of sensors on all links are given by defining the position of the sensor in the coordinate system of the link. The normal vector of each sensor is specified in the link's coordinate system.

### F.1.1 Link 0 : The base link

This link is the stationary vertical stand which supports the PUMA. Its actual shape is a cylinder, but it is approximated to be an extruded hexagonal shape. Figure F.1 shows the approximate shape of the link used for modelling. No sensors are placed on this link since it is stationary.

**Vertices**

| | | |
|---|---|---|
| 1 | ≡ | (−90, 0, −80) |
| 2 | ≡ | (−45, 78, −80) |
| 3 | ≡ | (45, 78, −80) |
| 4 | ≡ | (90, 0, −80) |
| 5 | ≡ | (45, −78, −80) |
| 6 | ≡ | (−45, −78, −80) |
| 7 | ≡ | (−90, 0, −740) |
| 8 | ≡ | (−45, 78, −740) |
| 9 | ≡ | (45, 78, −740) |
| 10 | ≡ | (90, 0, −740) |
| 11 | ≡ | (45, −78, −740) |
| 12 | ≡ | (−45, −78, −740) |

**Edges**

1–2
2–3
3–4
4–5
5–6
6–1
7–8
8–9
9–10
10–11
11–12
12–7
1–7
2–8
3–9
4–10
5–11
6–12

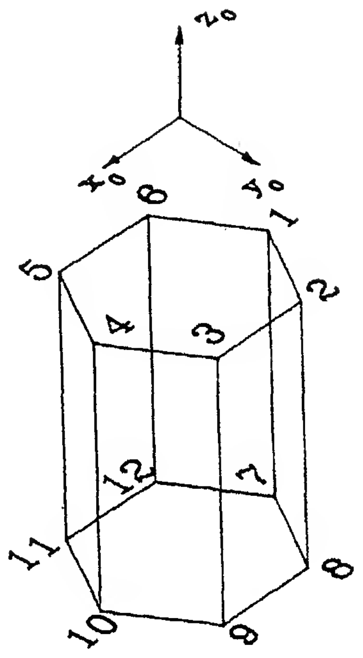


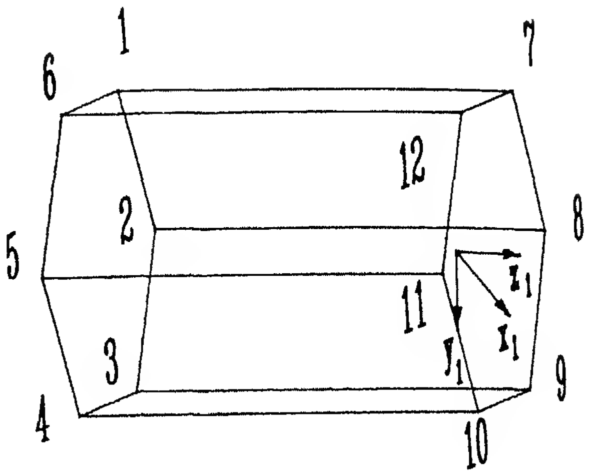
Figure F.1: Shape of Link '0' of PUMA for modelling

**Vertices**

| | | |
|---|---|---|
| 1 | ≡ | (0, −80, −65) |
| 2 | ≡ | (69, −40, −65) |
| 3 | ≡ | (69, 40, −65) |
| 4 | ≡ | (0, 80, −65) |
| 5 | ≡ | (−69, 40, −65) |
| 6 | ≡ | (−69, −40, −65) |
| 7 | ≡ | (0, −80, 200) |
| 8 | ≡ | (69, −40, 200) |
| 9 | ≡ | (69, 40, 200) |
| 10 | ≡ | (0, 80, 200) |
| 11 | ≡ | (−69, 40, 200) |
| 12 | ≡ | (−69, −40, 200) |

**Edges**

1–2
2–3
3–4
4–5
5–6
6–1
7–8
8–9
9–10
10–11
11–12
12–7
1–7
2–8
3–9
4–10
5–11
6–12



**Sensors located at**
(−80, 0, −65) with Normal [−10, 0, −10]
(80, 0, −65) with Normal [10, 0, −10]

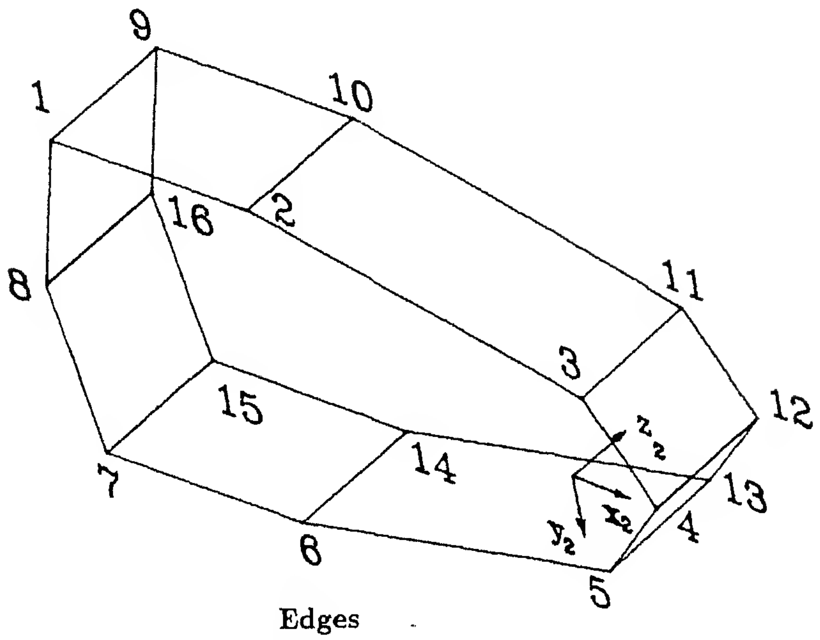
Figure F.2: Shape of Link '1' of PUMA for modelling

### F.1.2 Link 1

Again this body is approximated from a circular shape to a hexagonal one for convenience. This link, which is a horizontal cylinder, moves with movement of joint 1. Two sensors on this link have been placed for demonstration, but it is not mandatory because the freedom on this link is very restrictive. Figure F.2 shows the approximate shape of the link used for modelling.

### F.1.3 Link 2

The second link of the PUMA is of complex shape and modelling it merely with straight planes does not give its true shape. However, a shape which closely approximates the actual shape is given in Figure F.3 with the list of vertices and edges. Three sensors are placed towards the *elbow* side of the link to guard against obstacles.



**Vertices**

| | | |
|---|---|---|
| 1 | ≡ | (−612, −150, 50) |
| 2 | ≡ | (−412, −150, 50) |
| 3 | ≡ | (8, −75, 50) |
| 4 | ≡ | (68, 0, 50) |
| 5 | ≡ | (8, 75, 50) |
| 6 | ≡ | (−412, 150, 50) |
| 7 | ≡ | (−612, 150, 50) |
| 8 | ≡ | (−672, 0, 50) |
| 9 | ≡ | (−612, −150, 150) |
| 10 | ≡ | (−412, −150, 150) |
| 11 | ≡ | (8, −75, 150) |
| 12 | ≡ | (68, 0, 150) |
| 13 | ≡ | (8, 75, 150) |
| 14 | ≡ | (−412, 150, 150) |
| 15 | ≡ | (−612, 150, 150) |
| 16 | ≡ | (−672, 0, 150) |

**Edges**

1–2
2–3
3–4
4–5
5–6
6–7
7–8
8–1
9–10
10–11
11–12
12–13
13–14
14–15
15–16
16–9
1–9
2–10
3–11
4–12
5–13
6–14
7–15
8–16

**Sensors located at**

(8, −75, 150) with Normal [0, −10, 10]
(8, 75, 150) with Normal [0, 10, 10]
(8, 75, 50) with Normal [0, 10, 0]

Figure F.3: Shape of Link '2' of PUMA for modelling

### F.1.4 Link 3

The third link of PUMA is similar to link 2. Since this link has more freedom of movement, it is meaningful to install more sensors on this link. Six sensors are installed on this link at vantage points. The modelling features of link 3 are depicted in Figure F.4

### F.1.5 Link 4

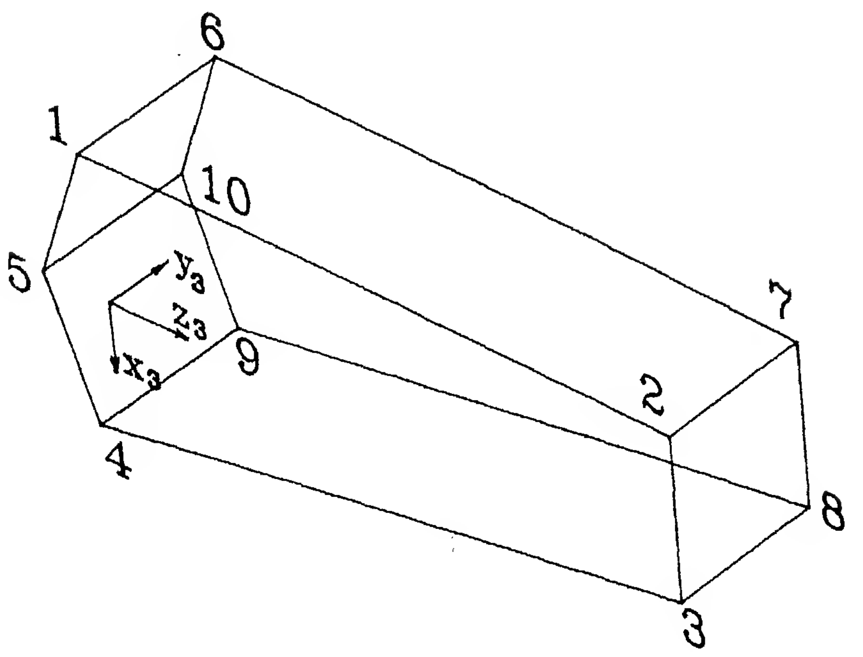
Link 4 of PUMA is small in size and is responsible for rotation of the hand vector. It is more or less round in shape and offers little scope for placement of sensors because of limited surface area. But since link 4 has high degree of orientability, it offers one of the most lucrative spots for placing sensors because it can *see* difficult areas easily. In the simulation program, four sensors have been placed on link 4. Figure F.5 describes the other details.

### F.1.6 Link 5

Although link 5 is needed for orientation, it does not form part of the outer body of the robot. It is hidden inside the robot's wrist and so is immaterial for modelling. Obviously, no sensors can be mounted when it is not visible from outside.

### F.1.7 Link 6

Strictly, link 6 is also not visible, but since link 6 is directly connected to the gripper, the gripper itself may be considered part of link 6. The gripper is modelled simply by considering it as a thick sheet of bent metal. Two sensors have been installed at the sides of the gripper.



| Vertices | | |
|---|---|---|
| 1 | ≡ | (−60, −35, −80) |
| 2 | ≡ | (−60, −35, 350) |
| 3 | ≡ | (25, −35, 350) |
| 4 | ≡ | (100, −35, −80) |
| 5 | ≡ | (20, −35, −110) |
| 6 | ≡ | (−60, 50, −70) |
| 7 | ≡ | (−60, 50, 350) |
| 8 | ≡ | (25, 50, 350) |
| 9 | ≡ | (100, 50, −80) |
| 10 | ≡ | (20, 50, −110) |

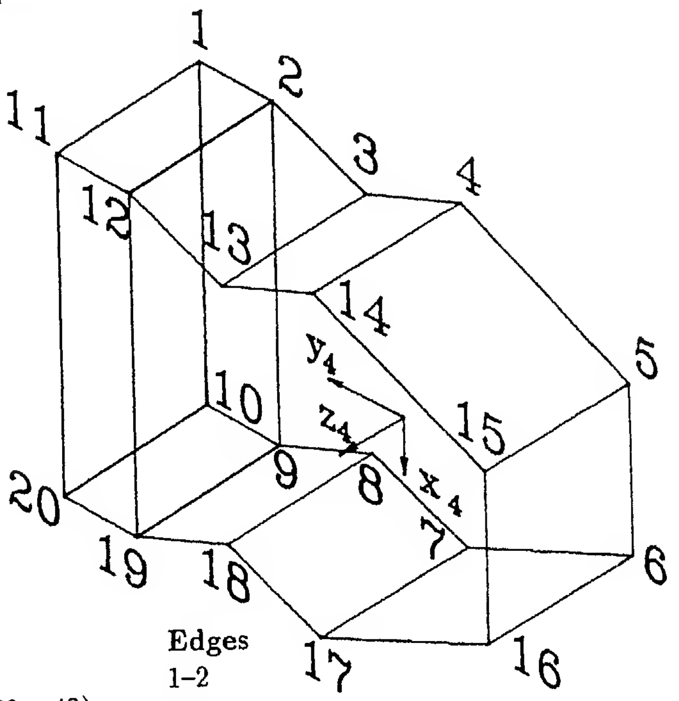
**Edges**

1–2
2–3
3–4
4–5
5–1
6–7
7–8
8–9
9–10
10–6
1–6
2–7
3–8
4–9
5–10

**Sensors located at**

| | | | |
|---|---|---|---|
| (−60, 10, −80) | with | Normal | [−10, 0, 0] |
| (100, 10, −80) | with | Normal | [10, 0, 0] |
| (−60, −35, 140) | with | Normal | [−10, 0, 0] |
| (−60, 50, 140) | with | Normal | [−10, 0, 0] |
| (62, −35, 140) | with | Normal | [10, −10, 0] |
| (62, 50, 140) | with | Normal | [10, 10, 0] |

Figure F.4: Shape of Link '3' of PUMA for modelling



**Vertices**

| | | |
|---|---|---|
| 1 | ≡ | $(-40, 80, -42)$ |
| 2 | ≡ | $(-40, 40, -42)$ |
| 3 | ≡ | $(-30, 30, -42)$ |
| 4 | ≡ | $(-40, 0, -42)$ |
| 5 | ≡ | $(-25, -40, -42)$ |
| 6 | ≡ | $(25, -40, -42)$ |
| 7 | ≡ | $(40, 0, -42)$ |
| 8 | ≡ | $(30, 30, -42)$ |
| 9 | ≡ | $(40, 40, -42)$ |
| 10 | ≡ | $(40, 80, -42)$ |
| 11 | ≡ | $(-40, 80, 42)$ |
| 12 | ≡ | $(-40, 40, 42)$ |
| 13 | ≡ | $(-30, 30, 42)$ |
| 14 | ≡ | $(-40, 0, 42)$ |
| 15 | ≡ | $(-25, -40, 42)$ |
| 16 | ≡ | $(25, -40, 42)$ |
| 17 | ≡ | $(40, 0, 42)$ |
| 18 | ≡ | $(30, 30, 42)$ |
| 19 | ≡ | $(40, 40, 42)$ |
| 20 | ≡ | $(40, 80, 42)$ |

**Edges**

1–2
2–3
3–4
4–5
5–6
6–7
7–8
8–9
9–10
10–1
11–12
12–13
13–14
14–15
15–16
16–17
17–18
18–19
19–20
20–11
2–12
9–19
5–15
6–16

**Sensors located at**

$(-40, 60, 0)$ with Normal $[-10, 0, 0]$
$(40, 60, 0)$ with Normal $[10, 0, 0]$
$(0, 0, -42)$ with Normal $[0, 0, -10]$
$(0, 0, 42)$ with Normal $[0, 0, 10]$

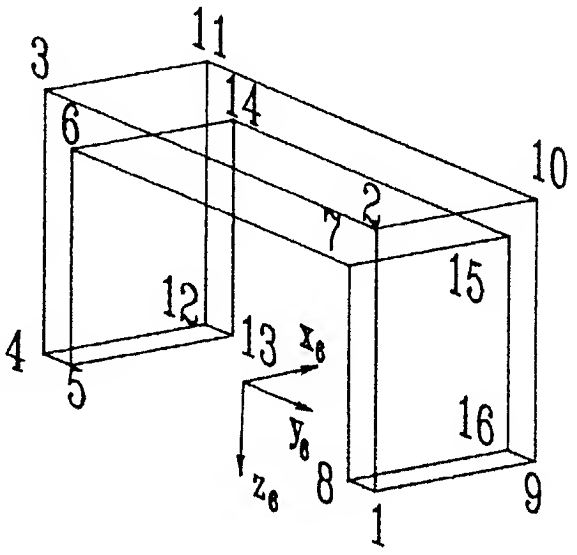
Figure F.5: Shape of Link '4' of PUMA for modelling

**Vertices**

| | | |
|---|---|---|
| 1 | ≡ | $(-40, 42, 50)$ |
| 2 | ≡ | $(-40, 42, 0)$ |
| 3 | ≡ | $(-40, -42, 0)$ |
| 4 | ≡ | $(-40, -42, 50)$ |
| 5 | ≡ | $(-40, -37, 50)$ |
| 6 | ≡ | $(-40, -37, 5)$ |
| 7 | ≡ | $(-40, 37, 5)$ |
| 8 | ≡ | $(-40, 37, 50)$ |
| 9 | ≡ | $(40, 42, 50)$ |
| 10 | ≡ | $(40, 42, 0)$ |
| 11 | ≡ | $(40, -42, 0)$ |
| 12 | ≡ | $(40, -42, 50)$ |
| 13 | ≡ | $(40, -37, 50)$ |
| 14 | ≡ | $(40, -37, 5)$ |
| 15 | ≡ | $(40, 37, 5)$ |
| 16 | ≡ | $(40, 37, 50)$ |

**Edges**

1–2
2–3
3–4
4–5
5–6
6–7
7–8
8–1
9–10
10–11
11–12
12–13
13–14
14–15
15–16
16–9
1–9
2–10
3–11
4–12
5–13
6–14
7–15
8–16



**Sensors located at**

$(0, -42, 25)$ with Normal $[0, -10, 0]$

$(0, 42, 25)$ with Normal $[0, 10, 0]$

Figure F.6: Shape of Link '6' of PUMA for modelling

# References

[1] Aronov, B., Fortune, S., Wilfong, G., "Minimum-Speed Motions," *The International Journal of Robotics Research*, vol. 10, no. 3, pp. 228-239 (June 1991).

[2] Ayache, N., Faugeras, O.D., "Maintaining Representations of the Environment of a Mobile Robot," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 6, pp. 804-819 (1989).

[3] Badaloni, S., Pagello, E., Sossai, C., "An Autonomous Mobile Robot in a Temporally Rich Domain : Some Considerations from the Standpoint of AI," In *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 672-682 (1990).

[4] Barraco, A., Xing, D.M., "Determination of Trajectories with Obstacle Avoidance," *Proceedings NATO ASI Series, Languages for Sensor-Based Control in Robotics*, Ed. U. Rembold and K. Hörmann, Springer-Verlag Berlin, pp. 362-388 (1987).

[5] Barraquand, J., Langlois, B., Latombe J.-C., "Robot Motion Planning with Many Degrees of Freedom and Dynamic Constraints," In *Robotics Research, The Fifth International Symposium*, Tokyo, Japan, pp. 435-444 (1989).

[6] Basu, A., Aloimonos, J., "Navigating in the Presence of Moving Obstacles," *Proceedings of the International Symposium on Intelligent Robotics*, Ed. M. Vidyasagar and M. Trivedi, Tata McGraw-Hill Publishing Company Ltd., New Delhi, pp. 242-253 (1991).

[7] Borenstein, J., Koren, Y., "Obstacle Avoidance with Ultrasonic Sensor," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 213-218 (1988).

[8] Borenstein, J., Koren, Y., "Real-time Obstacle Avoidance for Fast Mobile Robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1179-1187 (Sept/Oct 1989).

[9] Borenstein, J., Koren, Y., "The Vector Field Histogram — Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 278-288 (June 1991).

[10] Borenstein, J., Koren, Y., "Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 535-539 (August 1991).

[11] Brooks, R.A., "Symbolic Error Analysis and Robot Planning," *International Journal of Robotics Research*, vol. 1, no. 4, (1982).

[12] Brooks, R.A., "Planning Collision-free Motions for Pick-and-Place Operations," *International Journal of Robotics Research*, vol. 2, no. 4, (1983).

[13] Brooks, R.A., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems Man and Cybernetics*, vol. SMC-13, no. 3, pp. 190-197 (March/April 1983).

[14] Brooks, R.A., "Visual Map making for Mobile Robot," In *IEEE International Conference on Robotics and Automation*, IEEE, New York, pp. 824-829 (1985).

[15] Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14-23 (March 1986).

[16] Burger, W., Bhanu, B., "Qualitative Understanding of Scene Dynamics for Mobile Robots," *International Journal of Robotics Research*, vol. 9, no. 6, pp. 74-90 (December 1990).

[17] Canny, J., Reif, J., "New Lower Bound Techniques for Robot Motion Planning Problems," In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, IEEE, New York, pp. 49-60 (1987).

[18] Canny, J.F., *The Complexity of Robot Motion Planning*, Ph.D Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1987).

[19] Canny, J., Donald, B., "Simplified Voronoi Diagrams," *Discrete and Computational Geometry*, Springer-Verlag, New York, vol. 3, no. 3, pp. 219-236 (1988).

[20] Cassinis, R., Venuti, P., "Sonar Range Data Processing and Enhancement," In *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 168-177 (1990).

[21] Chatila, R., Laumond, J.P., "Position Referencing and Consistent World Modelling for Mobile Robots," In *IEEE International Conference on Robotics and Automation*, St. Louis, pp. 138-144 (1985).

[22] Cho, D.W., "Certainty Grid Representation for Robot Navigation by a Bayesian Method," *ROBOTICA*, vol. 8, Part 2, pp. 159-165 (1990).

[23] Ciliz, M.K., Isik, C., "Fuzzy Rule-based Motion Controller for an Autonomous Mobile Robot," *ROBOTICA*, vol. 7, Part 1, pp. 37-42 (1989).

[24] Cockroft, A.N., Lameyer, J.N.F., *A Guide to the Collision Avoidance Rules*, Stanford Maritime, London (1987).

[25] Cox, I.J., "Blanche : Position Estimation for an Autonomous Robot Vehicle," In *Proceedings of the IEEE/RSJ International Workshop on Robots and Systems (IROS'89)*, Tuksuba, Japan, pp. 432-439 (1989).

[26] Cox, I.J., Wilfong, G.T., *Autonomous Robot Vehicles*, Springer-Verlag, New York (1990).

[27] Crowley, J.L., "Dynamic World Modelling for an Intelligent Mobile Robot Using a Rotating Ultra-Sonic Ranging Device," *IEEE International Conference on Robotics and Automation*, Missouri, pp. 128-135 (1985).

[28] Crowley, J.L., "World Modelling and Position Estimation for Mobile Robot Using Ultrasonic Ranging," In *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, IEEE, New York, pp. 674-680 (1989).

[29] Denavit, J., Hartenberg, R.S., "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices," *J. Applied Mechanics*, vol. 22, pp. 215-221 (1955).

[30] Donald, B.R., *Error Detection and Recovery for Robot Motion Planning with Uncertainty*, Ph.D Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1987).

[31] Donald, B.R., "A Geometric Approach to Error Detection and Recovery for Robot Motion Planning with Uncertainty," *Artificial Intelligence*, vol. 37, no. 1-3, pp. 223-271 (1988).

[32] Donald, B.R., "Planning Multi-Step Error Detection and Recovery Strategy," *The International Journal of Robotics Research*, vol. 9, no. 1, pp. 3-60 (Feb. 1990).

[33] Durrant-White, H.F., "Consistent Integration and Propagation of Disparate Sensor Observations," *International Journal of Robotics Research*, vol. 6, no. 3, pp. 3-24 (1987).

[34] Elfes, A., "Sonar Based Real-World Mapping and Navigation," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 249-265 (June 1987).

[35] Elgazzar, S., *Efficient Solution for the Kinematic Positions for the PUMA 560 Robot*, NRC/ERB-973, National Research Council of Canada (December 1984).

[36] Elgazzar, S., *Efficient Solution for the Kinematic Velocities and Accelerations for the PUMA 560 Robot*, ERB-974/NRCC No. 25052, National Research Council of Canada (October 1985).

[37] Erdmann, M.A., "Using Backprojections for Fine-Motion Planning with Uncertainty," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 19-45 (Spring 1986).

[38] Faugeras, O.D., Hebert, M., "The Representation, Recognition, and Locating of 3-D Objects," *International Journal of Robotics Research*, vol. 5, no. 3, pp. 27-52 (Fall 1986).

[39] Faverjon, B., "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator," In *IEEE International Conference on Robotics and Automation*, Atlanta (March 1984).

[40] Faverjon, B., Tournassoud, P., "Object-level Programming of Industrial Robots," In *IEEE International Conference on Robotics and Automation*, San Francisco, pp. 1406-1411 (April 1986).

[41] Faverjon, B., Tournassoud, P., "A Practical Approach to Motion Planning for Manipulators with Many Degrees of Freedom," In *Robotics Research, The Fifth International Symposium*, Tokyo, Japan, pp. 425-433 (1989).

[42] Ferrari, C., Chemello, G., "Coupling Fuzzy Logic Techniques with Evidential Reasoning for Sensor Data Interpretation," in *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 965-971 (1990).

[43] Foulloy, L., Mauris, G., "An Ultrasonic Fuzzy Sensor," *Proc. Int. Conf. Robot Vision and Sensory Controls*, pp. 161-170 (February 1988).

[44] Fu, K.S., Gonzalez, R.C., Lee, C.S.G., *Robotics, Control Sensing, Vision, and Intelligence*, McGraw-Hill Book Company, Singapore (1987).

[45] Fujimura, K., Samet, H., "A Hierarchical Strategy for Path Planning Among Moving Obstacles," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 61-69 (Feb. 1989).

[46] Geier, G.J., Cabak, A., Sieh, L., "Design of an Integrated Navigation System for Robotic Vehicle Applications," *Navigation*, vol. 34, no. 4, pp. 325-336 (1987-1988).

[47] Grimson, W.E.L., Lozano-Pérez, T., "Localizing Overlapping Parts by Searching the Interpretation Tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 4, pp. 469-482 (July 1987).

[48] Grimson, W.E.L., "Sensing Strategies for Disambiguating Motion Among Multiple Objects in Known Poses," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 4, pp. 196-213 (December 1986).

[49] Griswold, N.C., Eem, J., "Control for Mobile Robots in the Presence of Moving Objects," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 263-268 (April 1990).

[50] Guibas, L., Sharir, M., Sifrony, S., "On the General Motion Planning Problem with Two Degrees of Freedom," In *Proceedings of the 4th Annual Symposium on Computational Geometry*, ACM, New York, pp. 289-298 (1988).

[51] Gupta, K.K., "Planning Collision-free Motions for a Manipulator Arm : Link by Link," *Proceedings of the IASTED International Symposium Robotics and Manufacturing*, pp. 192-196 (1989).

[52] Hamakawa, R., "One General Navigation Strategy for Intelligent Mobile Robots," In *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 823-833 (1990).

[53] Harrington, S., *Computer Graphics : A Programming Approach*, McGraw Hill Book Company, Singapore (1983).

[54] Henderson, T., Silcrat, E., "Logical Sensor Systems," *Journal of Robotics Systems*, vol. 1, no. 2, pp. 169-193 (1984).

[55] Hoffmann, C.M., "The Problems of Accuracy and Robustness in Geometric Computation," *IEEE Computer*, vol. 22, no. 3, pp. 31-41 (1989).

[56] Hogan, N., "Impedance Control : An Approach to Manipulation," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 1-7 (March 1985).

[57] Hogan, N., "Impedance Control : An Approach to Manipulation," In *American Control Conference*, (June 1984).

[58] Horowitz, E., Sahni, S., *Fundamentals of Data Structures*, CBS Publishers and Distributors, (Indian Edition) New Delhi (1983).

[59] Ikegami, T., Kato, J., Ozono, S., "Sensor Data Integration Based on the Border Distance Model," In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, IEEE/RSJ*, New York, pp. 32-39 (1989).

[60] Jacobs, P., Canny, J., "Planning Smooth Paths for Mobile Robots," In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, Washington, pp. 2-7 (1989).

[61] Kanades, P.N., Langen, A., Warnecke Hans-Jürgen, "A Combined Generalized Potential Fields/Dynamic Path Planning Approach to Collision Avoidance for a Mobile Autonomous Robot Operating in a Constrained Environment," In *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 145-154 (1990).

[62] Kant, K., Zucker, S., "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition," *International Journal of Robotics Research*, vol. 5, no. 3, pp. 72-89 (1986).

[63] Kedem, K., Sharir, M., "An Automatic Motion Planning System for a Convex Polygonal Mobile Robot in 2-Dimensional Polygonal Space," In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pp. 329-340 (1988).

[64] Khatib, O., Le Maitre, J.-F., "Dynamic Control of Manipulators operating in a complex environment," In *Proceedings Third International CISM-IFToMM Symposium*, Udine, Italy, pp. 267-282 (September 1978).

[65] Khatib, O.,"Real-time obstacle avoidance for manipulators and mobile robots," *International Journal Robotics Research*, vol. 5, no. 1, pp. 90-98 (Spring 1986).

[66] Kim, M.S., "Motion Planning with Geometric Models," Ph.D Thesis, Pohang Institute of Science and Technology (1989).

[67] Koditschek, D.E., "Exact Robot Navigation by Means of Potential Functions : Some Topological Considerations," In *IEEE International Conference on Robotics and Automation*, Raleigh, NC, pp. 1-6 (March 1987).

[68] Koditschek, D.E., "Robot Planning and Control via Potential Functions," In *The Robotics Review 1* Oussama Khatib, John J. Craig and Tomás Lozano-Pérez, Eds. MIT Press, Cambridge, Mass., pp. 349-368 (1989).

[69] Kondo, K., "Motion Planning with Six Degrees of Freedom by Multistage Bidirectional Heuristic Free-Space Enumeration," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 267-277 (June 1991).

[70] Krogh, B.H., "Feedback obstacle avoidance control," *21st Allerton Conference*, University of Illinois (October 1983).

[71] Kröse, Ben J.A., Dondorp, E., "A Sensor Simulation System for Mobile Robots," In *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 641-649 (1990).

[72] Kuc, R., Seigel, M.W., "Physically-Based Simulation Model for Acoustic Sensor Robot Navigation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 6, pp. 766-778 (1987).

[73] Kuc, R., Viard, V.B., "A Physically Based Navigation Strategy for Sonar-Guided Vehicles," *International Journal of Robotics Research*, vol. 10, no. 2, pp. 75-87 (April 1991).

[74] Kuc, R., Siegel, M.W., "Physically based simulation model for acoustic sensor robot navigation," *IEEE Trans. on Patt. Anal. and Machine Intell.*, vol. PAMI-9, no. 6, pp. 766-778 (1987).

[75] Larsen, P.M., "Industrial Applications of Fuzzy Logic Control", *International Journal of Man-Machine Studies*, vol. 12, pp. 3-10 (1980).

[76] Laumond, J., "Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints," In *Intelligent Autonomous Systems*, pp. 346-354 (1986).

[77] Lee, J., Bein, Z., "Collision-free Trajectory Control For Multiple Robots Based on Neural Optimization Network," *ROBOTICA*, vol. 8, no. 3, pp. 185-194 (1990).

[78] Liu, Y.-H., Kuroda, S., Naniwa, T., Noborio, H., Arimoto, S., "A Practical Algorithm for Planning Collision-Free Coordinated Motion of Multiple Mobile Robots," *IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona, pp. 1427-1432 (1989).

[79] Lozano-Pérez T., Wesley, M.A., "An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560-570 (October 1979).

[80] Lozano-Pérez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems Man and Cybernetics*, vol. SMC-11, no. 10, pp. 681-698 (October 1981).

[81] Lozano-Pérez, T., "Spatial Planning : A Configuration Space Approach," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 108-120 (February 1983).

[82] Lozano-Pérez, T., "Robot Programming," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 821-841 (July 1983).

[83] Lozano-Pérez, T., "A Simple Motion Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 224-238, (June 1987).

[84] Lozano-Pérez, T., Foreword in *Autonomous Robot Vehicles* ed. Ingemar J. Cox and Gordon T. Wilfong, Springer-Verlag, New York (1990).

[85] Lozano-Pérez T., Brooks, R.A., *An Approach to Automatic Robot Programming*, MIT A.I. memo no. 842, Massachusetts Institute of Technology (April 1985).

[86] Lozano-Pérez, T., Mason, M.T., Taylor, R.H., "Automatic Synthesis of Fine-Motion Strategies for Robots," *Proceedings of the First International Symposium on Robotics Research*, Bretten-Woods, NH, pp. 65-96. Also reprinted in *International Journal of Robotics Research*, vol. 3, no. 1, pp. 3-24 (1983).

[87] Lozano-Pérez, T., Taylor, R.H., "Geometric Issues in Planning Robot Tasks," In *Robotics Science* ed. Michael Brady, MIT Press, Cambridge, Massachusetts, pp. 227-261 (1989).

[88] Luh, J.Y.S., Walker, M.W., Paul, R.P., "On-line Computational Scheme for Mechanical Manipulators," *Trans. ASME, J. Dynamic Systems, Measurements and Control*, vol. 120, pp. 69-76 (1980).

[89] Lumelsky, V.J., Stepanov, A.A., "Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment," *IEEE Transactions on Automatic Control*, vol. AC-31, no. 11, pp. 1058-1063 (1986).

[90] Lumelsky, V.J., "Dynamic Path Planning for a Planar Articulated Robot Arm Moving Amidst Unknown Obstacles," *Automatica, Journal International Federation of Automatic Control (IFAC)*, (September 1987).

[91] Lumelsky, V.J., "Effect of Kinematics on Dynamic Path Planning for Planar Robot Arms Moving Amidst Unknown Obstacles," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 207-223 (1987).

[92] Lumelsky, V.J., Stepanov, A.A., "Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape," *Algorithmica*, vol. 2, no. 4, pp. 403-440 (1987).

[93] Lumelsky, V., Skewis, T., "A Paradigm for Incorporating Vision in the Robot Navigation Function," In *IEEE International Conference on Robotics and Automation*, IEEE, New York, pp. 734-739 (1988).

[94] Lumelsky, V.J., Sun, K., "A Unified Methodology for Motion Planning with Uncertainty for 2D and 3D Two-Link Robot Arm Manipulators," *Int. J. Robotics Research*, vol. 9, no. 5, pp. 89-104 (1990).

[95] Mamdani, E.H., "Process Control Using Fuzzy Logic," *Designing for Human-Computer Communication*, Academic Press, London (1983).

[96] Mason, M.T., "Automatic Planning of Fine Motions : Correctness and Completeness," *IEEE International Conference on Robotics and Automation*, Atlanta, pp. 492-503 (1984).

[97] Maybeck, P.S., *Stochastic Models, Estimation, and Control*, vol. 1, Academic Press, Boston (1979).

[98] McMillan, J.C., "An Integrated System for Land Navigation," *Jour. Instit. Navigation*, vol. 34, no. 1, pp. 43-63 (1987).

[99] Merriam-Webster, *Webster's Ninth New Collegiate Dictionary*, Merriam-Webster, Springfield, Massachusetts (1985).

[100] Moravec, H.P., "Sensor Fusion in Certainty Grids for Mobile Robots," *Proceedings NATO ASI Series, Sensor Devices and Systems for Robotics*, Ed. A. Casals, Vol. F52, Springer-Verlag Berlin, pp. 254-276 (1989).

[101] Novák, Vilém, *Fuzzy Sets and their Applications* Adam Hilger, Bristol (1989).

[102] Ó'Dúnlaing, C., "Motion Planning with Inertial Constraints," *Algorithmica*, vol. 2, no. 4, pp. 431-475 (1987).

[103] Parnis, P.J., Drazan, P.J., "Recognition of Unreliable Ultrasonic Data in a Robotic Environment," *ROBOTICA*, no. 7, Part 3, pp. 223-229 (1989).

[104] Polaroid Corp., *Ultrasonic range finders*, Ultrasonic ranging and marketing division, Cambridge, Mass (1982).

[105] Reif, J., Sharir, M., "Motion Planning in the Presence of Moving Obstacles," In *Proceedings of the 26th Symposium on Foundations of Computer Science*, IEEE, New York, pp. 144-154 (1985).

[106] Renaud, M., *Contribution à l' étude de la modelisation des systèmes mécaniques articulés*, Thèse de Docteur-Ingénieur, Univ. Toulouse (December 1976).

[107] Rimon, E., Koditschek, D.E., "Exact Robot Navigation Using Cost Functions : The Case of Spherical Boundaries in $E^n$," In *IEEE International Conference on Robotics and Automation*, Philadelphia, PA (April 1988).

[108] Scharf, E.M., Mandič, Mamdani, E.H., "A Self-Organizing Algorithm for the Control of A Robot Arm," *International Journal of Robotics and Automation*, vol. 1, no. 1, pp. 33-41 (1986).

[109] Schilling, R.J., *Fundamentals of Robotics, Analysis and Control*, Prentice-Hall, Englewood Cliffs, New Jersey (1990).

[110] Schneiter, J.L., "An Objective Tactile Sensing Strategy for Object Recognition and Localization," In *IEEE International Conference on Robotics and Automation*, San Francisco, pp. 1262-1267 (1986).

[111] Schwartz, J.T., Sharir, M., "On the Piano Movers' Problem : III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers," *International Journal of Robotics Research*, vol. 2, no. 3, pp. 46-75 (1983).

[112] Schwartz, J.T., Sharir, M., "On the Piano Movers' Problem : II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," *Advances in Applied Mathematics*, vol. 4, pp. 298-351 (1983).

[113] Schwartz, J.T., Sharir, M., "On the Piano Movers' Problem : I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345-398 (1983).

[114] Schwartz, J.T., Sharir, M., "On the Piano Movers' Problem : V. The Case of a Rod Moving in Three-Dimensional Space Amidst Polyhedral Obstacles," *Communications on Pure and Applied Mathematics*, vol. 37, pp. 815-848 (1984).

[115] Sharir, M., Ariel-Sheffi, E., "On the Piano Movers' Problem : IV. Various Decomposable Two-Dimensional Planning Problems," *Communications on Pure and Applied Mathematics*, vol. 37, pp. 479-493 (1984).

[116] Singh, S.K., Leu, M.C., "Manipulator Motion Planning in the Presence of Obstacles and Dynamic Constraints," *The International Journal of Robotics Research*, vol. 10, no. 2, pp. 171-187 (April 1991).

[117] Smith, R., Cheeseman, P., "On the Representation and Estimation of Spatial Uncertainty," *International Journal of Robotics Research*, vol. 5, no. 4, pp. 56-68 (Winter 1986).

[118] Smith, R., Self, M., Cheeseman, P., "Estimating Uncertain Spatial Relationships in Robotics," *Uncertainty in Artificial Intelligence* Eds. Lemmer/Kanal, Elsevier Science Publishers, vol. 2, pp. 435-461 (1988).

[119] Soetadji, T., "A Cube-Based Approach to Find-Path for an Autonomous Mobile Robot," *Proceedings NATO ASI Series, Languages for Sensor-Based Control in Robotics*, Ed. U. Rembold and K. Hörmann, Springer-Verlag Berlin, pp. 558-588 (1987).

[120] Sood, D., Repko, M.C., Kelley, R.B., "Design and Implementation of a Multi-Sensor Robot System for Printed Circuit Board Insertion," *IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona, pp. 377-382 (1989).

[121] Sun, K., Lumelsky, V., "Simulating Sensor-Based Robot Motion Amongst Unknown Obstacles," *Proceedings NATO ASI Series, Languages for Sensor-Based Control in Robotics*, Ed. U. Rembold and K. Hörmann, Springer-Verlag Berlin, pp. 451-479 (1987).

[122] Sun, K., Lumelsky, V.J., "Computer simulation of sensor-based robot collision avoidance in an unknown environment," *ROBOTICA*, vol. 5, Part 4, pp. 291-302 (1987).

[123] Sun, K., Lumelsky, V.J., "Motion Planning with Uncertainty for a 3d Cartesian Robot Arm," *Robotics Research, The Fifth International Symposium*, Tokyo, Japan, pp. 417-424 (1989).

[124] Takahashi, O., Schilling, R.J., "Motion Planning in a Plane Using Generalized Voronoi Diagrams," *IEEE Transactions on Robotics and Automation*, vol. RA-5, no. 2, pp. 143-150 (1989).

[125] Tarn, T.J., Bejczy, A.K., "Software Elements," In *International Encyclopedia of Robotics : Applications and Automation* Eds. Richard C. Dorf and Shimon Y. Nof, John Wiley & Sons, vol. 3, pp. 1608-1626 (1988).

[126] Taylor, R.H., "The Synthesis of Manipulator Control Programs from Task-Level Specifications," *Technical Report AIM-282*, Stanford University, Artificial Intelligence Laboratory (1976).

[127] Thomas, G.B., *Calculus and Analytic Geometry*, Addison-Wesley Publishing Company, Reading Massachusetts (1974).

[128] Torrieri, D.J., "Statistical Theory of Passive Location Systems," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-20, no. 2, pp. 183-198 (1984).

[129] Tuijnman, F., Kröse, Ben J.A., "Neural Networks for Collision Avoidance between Autonomous Mobile Robots," in *Proc. of Intelligent Autonomous Systems, Amsterdam*, pp. 407-417 (1990).

[130] Udapa, S., "Collision Detection and Avoidance in Computer Controlled Manipulators," In *Fifth International Joint Conference on Artificial Intelligence*, Cambridge (1977).

[131] *User's Guide to VAL : Programming Manual*, Kawasaki Heavy Industries Ltd, Robot Division (February 1987).

[132] Volpe, R., Khosla, P., "A Strategy for Obstacle Avoidance and Approach Using Super Quadrics Potential Functions," In *Robotics Research, The Fifth International Symposium*, Tokyo, Japan, pp. 445-452, (1989)

[133] Wenger, P., Chedmail, P., "Ability of a Robot to Travel Through its Free Work Space in an Environment with Obstacles," *The International Journal of Robotics Research*, vol. 10, no. 3, pp. 214-227 (June 1991).

[134] Yap., C.K., "Algorithmic Motion Planning," In *Advances in Robotics* editors J.T. Schwartz and C.K. Yap, vol. 1, Lawrence Erlbaum Associates (1987).

[135] Zadeh, L.A., "Fuzzy Sets," *Information Control*, vol. 8, pp. 338-353 (1965).

[136] Zadeh, L.A., "Outline of a New Approach to the Analysis of Complex System and Decision Processes," *IEEE Trans. on System Man and Cybernetics*, vol. SMC-3, no. 1, pp. 28-44 (1973).

[137] Zadeh, L.A., "The Concept of a Linguistic Variable and its Application to Approximate Reasoning I, II, III," *Information Science*, vol. 8, pp. 199-257; pp. 301-57; *ibid* vol. 9, pp. 43-80 (1975).

[138] Zhu, D., Latombe, J.-C., "New Heuristic Algorithms for Efficient Heirarchical Path Planning," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 9-20 (Feb 1991).

[139] Zhu, Q., "Hidden Markov Model for Dynamic Obstacle Avoidance of Mobile Robot Navigation," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 390-397 (June 1991).

# Index